LEVERAGING DISTRIBUTION AND HETEROGENEITY IN ROBOT SYSTEMS ARCHITECTURE

A Thesis
Presented to
The Academic Faculty

by

Keith J. O'Hara

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
December 2011

LEVERAGING DISTRIBUTION AND HETEROGENEITY IN ROBOT SYSTEMS ARCHITECTURE

Approved by:

Professor Tucker Balch, Adviser
College of Computing
Georgia Institute of Technology

Professor Henrik Christensen
College of Computing
Georgia Institute of Technology

Professor Mark Guzdial
College of Computing
Georgia Institute of Technology

Professor Karsten Schwan
College of Computing
Georgia Institute of Technology

Professor Gaurav Sukhatme
Computer Science Department
University of Southern California

Date Approved: July 29, 2011

To my family and my teachers.

ACKNOWLEDGEMENTS

Earning a doctorate is much like running a marathon, so they say. Both take lots of training, tend to be painful, last forever, and result in feeling great. And since I consider myself more of a sprinter than a long-distance runner, without a wonderful group of people, I wouldn't have finished my marathon.

First, I'd like to thank my committee for their guidance, thoughtfulness, and insight. All five members are evidence that great scientists can also be good people. Specifically, I'd like to thank Tucker for being such a strong supporter and advocate. Many teachers and mentors have guided my academic and intellectual journey, and I am forever grateful. I'm also grateful for the Intel Foundation's generous fellowship.

A big thank you to my collaborators at Georgia Tech and beyond: Ben Axelrod, Victor Bigio, Doug Blank, Eric Dodson, Can Evarli, Deepak Kumar, Ripal Nathuji, Jim Perkins, Jay Summet, Shaun Whitt, and the Borg Lab. A special nod to Daniel Walker.

I'd like to thank Rowan University for providing me with the opportunity to begin studying computer science, and my colleagues at Bard College for allowing me to pay it forward (and for their support and patience as I finished this dissertation).

We met some great people in Atlanta who made our stay fun as well as productive. Thanks to the Powers, Marlowes, Emelys, Masts, Bob and Kaitlin for all the great times – whole pumpkins or not.

I dedicated this dissertation to my family, their influence was immeasurable and their support was priceless.

And to Mary, thanks for your patience and unflinching confidence.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Like computer architects, robot designers must address multiple, possibly competing, requirements by balancing trade-offs in terms of processing, memory, communication, and energy to satisfy design objectives. However, robot architects currently lack the design guidelines, organizing principles, rules of thumb, and tools that computer architects rely upon. This thesis takes a step in this direction, by analyzing the roles of heterogeneity and distribution in robot systems architecture.

Robots live in the real world, sensing and effecting external physical phenomena. This leads to a key consideration that is sometimes lost in traditional computing system design. Where is the robot computing system located in physical space? This consideration amplifies the role distribution plays in robot system design. The allocation and organization of the sensing, computing, actuation, energy, communication resources throughout physical space is at the core of distributed robot systems architecture. The physical distribution of resources let's us exploit the locality of the particular task in a similar manner as the design of a memory subsystem let's us exploit the locality of a computational problem.

This thesis takes a systems architecture approach to the design of robot systems, and in particular, investigates the use of distributed, heterogeneous platforms to exploit locality in robot systems design. We show how multiple, distributed heterogeneous platforms can serve as general purpose robot systems for three distinct domains with different design objectives: increasing availability in a search and rescue mission, increasing flexibility and ease-of-use for a personal educational robot, and decreasing the computation and sensing resources necessary for navigation and foraging tasks.

CHAPTER I

INTRODUCTION

Just as special purpose computers and mainframes grew into the general purpose personal computers we use everyday, special purpose industrial robots are evolving into more general purpose *personal robots*. Robot systems have been proposed for a wide variety of tasks, from canonical robot tasks such as autonomous navigation[53] to providing a motivating context for education[77]. As robots become more capable and universal, their applications are less well-defined or even unknown at design time. We will have to design robots for classes of tasks rather than specific applications. Having guidelines for how to best organize, interface, and implement robot systems and reason about trade-offs, as we do in computer architecture[45], will become crucial for success.

This thesis takes a systems architecture approach to the design of robot systems, and in particular, understanding the use of distributed robot systems to achieve design objectives. The IEEE Standard 1471-2000[1] defines *Systems Architecture* as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution." Likewise, we define *Robot Systems Architecture* as "the fundamental organization of a robot system, embodied in its computation, communication, sensing, and actuation resources, their relationships to each other and the environment, and the principles governing its design and evolution."

It's quite natural in robot architecture to decompose a task into loosely coupled subtasks in service of more effective design, development, maintenance, and reuse, but it stops at the level of software. This approach goes beyond "Robot Software Architecture" which typically assumes the hardware necessary for the task is available and the details of the hardware are abstracted away. A robot software architecture (like subsumption[15], RCS[3], Aura[6]) gives the roboticist a set of principles, guidelines, and tools to compose a robot's software

[1] Recommended Practice for Architectural Description of Software-Intensive Systems

system. Robot systems architecture puts the hardware and software on equal footing in terms of emphasis and abstraction. We are concerned with the organization of hardware resources, and to some degree the environment, since the interface to the physical world is such a crucial part of any robot system, and particularly in exploiting the locality of the particular robot task. Modern advances in wireless communication have enabled a new level of composability in terms of hardware, allowing us to repurpose hardware analogously to how we reprogram software.

1.1 On Problem, Platform, and Process

Because task and system, or problem and platform, have traditionally been so tightly coupled in robot system design, the distinction between robot system architecture and task can become murky. Firstly, a robot task is the *problem* to be solved independent of any physical robot *platform*. The problem itself might impose some limitations on the platform (system architecture). The task might demand certain a sensor range or a particular physical size of the robot for the task to be achieved. For example, Donald[29] and O'Kane[74] look to understand the fundamental information requirements of specific robot tasks.

Secondly, independent of any task we can also talk about the platform, a robot system architecture – the organization of a robot system, embodied in its computation, communication, sensing, and actuation resources, their relationships to each other and the environment, and the principles governing its design and evolution. Physical limitations for example on motor speed and communication bandwidth can be examined independently of the task. For example, Xue and Kumar[107] derive network capacity laws for wireless sensor networks using different communication models.

Thirdly, there is the physical instantiation of a robot's systems architecture performing some robot task – the process. If we design a robot system architecture for a larger class of tasks, the platform will behave differently depending on the particular process.

Work like Parker et al.[78] looks to take a task description (problem) and output a robot design (platform). If the platform is designed specifically for a single problem, there is little difference between the process and the platform. But if you are designing a system for a set

of tasks, some of which are unknown at design-time, this approach is insufficient. Rather than fully automating the design process, we need a set of principles to guide the design. These principles can be discovered not only from problems and platforms, but from robot processes.

1.2 Distribution and Heterogeneity

Distribution is primarily employed in the design of robot systems for reasons of fault-tolerance and parallel speed-up. However, it also manifests itself as an implementation detail when fielding real systems. This thesis explores, through the design, implementation, and experimentation of heterogeneous distributed robots systems, the efficacy of distributed robot architecture to address other design criteria including: user-interface, cost, energy, flexibility, and reliability.

Robot systems are increasingly built as large distributed systems. Robot systems are built in a distributed fashion for both essential and incidental reasons. This leads us to our first way of classifying the role of distribution in robot systems.

- **Distribution is Essential** – The fact the robot system has multiple networked components is paramount in its design. For example, consider a team of unmanned aerial vehicles providing situational awareness. The multiplicity of vehicles is necessary to satisfy the performance and fault tolerance objectives.

- **Distribution is Incidental** – The fact that the robot software system is distributed across multiple machines is only an implementation detail and often an afterthought. Typically, the systems are distributed to allow off-loading of computation for practical performance reasons or to provide a remote user interface during development.

The use of distribution in robot systems is not new in itself – in fact, just the opposite, it is ubiquitous. Practically all real, fielded, robot systems are built as distributed systems; however, the distributed systems aspects of robot systems are often seen as ancillary and overlooked as "implementation details" rather than being a fundamental problem. Opportunities are lost by considering distribution too late in the development cycle.

The first argument this thesis puts forward is that all robot systems are inevitably going to be distributed systems, and the earlier this is taken into consideration the better.

In both the essential and incidental scenarios, distribution let's us exploit the locality of the robot task by placing the hardware resources close to where the robot computation needs to take place. Moreover, by using a collection of specialized platforms in a coordinated fashion, we can further exploit the locality of a task. Typical approaches that use a collection of identical platforms for parallel speed-up and fault tolerance through redundancy do not exploit this fact. This leads us to our second way of classifying distributed robot systems, by their composition. Is the robot system homogeneous or heterogeneous? Are all the hardware platforms identical, or near identical, or are the resources allocated in an asymmetric manner?

- **Homogeneous Composition** – The entities in the distributed robot systems are largely identical. Many multi-robot systems rely on multiple homogeneous platforms to perform tasks in a parallel or fault-tolerant fashion. The notion that we can achieve reliable, sophisticated performance from a multitude of unreliable, simple platforms is at work here.

- **Heterogeneous Composition** – The entities in the distributed robot system have differences in their hardware. For example, different robots might have different sensing or computational resources. Many systems that rely on distribution incidentally are highly heterogeneous. For example, a system might perform computational offloading for practical performance reasons or to provide a remote user interface.

Just as we strive to achieve reliable performance from a multitude of unreliable, simple homogeneous platforms, we can also aim to achieve generality from a multitude of specialized platforms.

The second argument this thesis offers is that robot systems can be constructed from a collection of specialized robots that are both sufficiently general to solve the task and sufficiently specialized to exploit the task's locality.

Dudek et al.[30] introduce composition as a dimension of their taxonomy of multi-robot

systems. Their taxonomy is used to classify some hypothetical multi-robot systems and 20 robot collectives from the research literature. They describe the robot collectives using a variety of dimensions including composition, team size, reconfigurability, communication range, cost, topology, and computation capability. Although these dimensions are useful in describing the team from a communication and coordination point of view, they largely ignore the capabilities of the individual platforms (e.g. the sensors and actuators). In addition, their technique for classifying communication bandwidth, computing capability, and composition is a bit too discrete and coarse for our needs.

This thesis argues that opportunities are lost by current practices of 1) considering distribution too late in the development cycle as in the case of *Incidental Distribution* or 2) by using distribution in situations of highly symmetric, homogeneous platforms as in the case of *Homogeneous Composition*.

Through the exposition of three robot systems that exploit heterogeneity and distribution in novel ways, this thesis offers not only lessons learned, but some general design guidelines for using distributed robot systems to achieve various design objectives. The principal contributions of this work include:

- **Using heterogeneity and distribution to exploit the spatial locality of the path planning, coordination and foraging tasks**.

  A technique is developed for performing path planning, coverage, and foraging using a system of heterogeneous robots. The locality of the path planning and coverage tasks is exploited by using a minimal, immobile sensor network, reducing the mobility and coordination resources needed by the mobile robots. Moreover, the resources necessary for the mobile robots (e.g. specific sensors or effectors) application are contained on the mobile robots lowering costs and providing flexibility. Empirical evidence shows the utility of the technique in tasks such as navigation and foraging. Experiments include simulation and some of the largest robot experiments in this domain.

- **Using heterogeneity and distribution to effectively manage energy in robot teams.**

  A system-level model for characterizing the energy behavior of distributed robot software systems is developed and applied to a multi-robot search and rescue mission. Distributed computing mechanisms are leveraged not only to speed-up computation, but to prolong the lifetime of the team. Experimental results using emulation of real robot software is used and the lifetime of the system is extended by 57%.

- **Using heterogeneity and distribution to create an inexpensive, but pedagogically rich educational robot platform.**

  A distributed robot system for teaching introductory computing is developed to maximize ease-of-use and pedagogical flexibility while minimizing cost. In addition, we assess the utility of personal robots in a multi-robot context for teaching computing concepts. Over four thousand of these robots have been deployed in over a variety of institutions.

1.3 Detailed Contributions

Distributed Navigation and Foraging [68, 69, 70, 71, 73]

- A path planning technique using a minimal embedded network rather than using traditional mapping and planning is designed and implemented.

- A technique for supporting multi-robot foraging is designed, implemented, and analyzed in terms of sensitivity to deployment and environmental conditions.

- The path planning algorithm is implemented on a real multi-robot system and the quality of the resulting paths is investigated.

- Evidence is provided that a mobile robot can effectively use the Gnats' paths in static and dynamic environments.

Energy-Aware Search and Rescue[72]

- A system-level model of software energy behavior and platform energy characteristics is developed.
- Software allocation and connectivity decisions are shown to prolong the lifetime of a robot team.

Personal Robots for Education[8, 97]

- A personal, distributed robot system for teaching introductory computing is designed and implemented.
- Evidence is provided for how a multi-robot system can aid in the learning of computing concepts.
- Evidence is presented that students are able to apply shortest path algorithms to navigation problems effectively after learning about the Gnats algorithms.

Robot Systems Architecture[67]

- The principles of locality and balanced computer architectures are applied to the study of robot system architecture.
- "Robot Design Constellations" are introduced as a means to explore design trade-offs and support comparison of robot platforms.

CHAPTER II

RELATED WORK

2.1 Embedded Networks

Xue and Kumar[107] explore how network capacity scales in wireless sensor networks using different communication models. Gupta and Kumar[43] use percolation theory to find what transmission power a sensor network should use to achieve connectivity with probability one. Shakkottai et al.[93] explore how unreliable nodes impact the connectivity and coverage of the network. Rachlin et al.[84, 85] cast the sensor network in an information theoretic light, and employ a coding theory method to the sensing capacity problem.

The pervasive computing community is interested in using embedded networks to support human activities. The Active Badge Location System [105] uses an embedded network for tracking people in a building. The Cricket system[82] from MIT is also used for tracking people in indoor settings. Also from the pervasive computing community, Kirsch and Starner [50] present the Locust platform specifically for supporting wearable computing in tasks such as messaging and localized storage.

Wireless sensor networks have been used to monitor outdoor environments[98]. For instance, detecting structural damage in buildings[19] and monitoring the behavior of glaciers [59]. The motes[24] and scatterweb[91] platforms are some of the most popular research platforms.

Mobile robots have also been used to support embedded networks. Lamarca uses mobile robots to continually calibrate a sensor network [54]. Rahimi presents an approach for power harvesting in sensor networks by exploiting mobility [86]. Corke uses a UAV to deploy and maintain the connectivity of a sensor network [22]. The use of mobile nodes as data mules often has benefits it terms of energy[104][18] and overall throughput, particularly in disconnected networks[58][109].

2.2 Mobile Robots

2.2.1 Mobile Robot Fundamentals

Donald[29] puts forth an approach based on "information invariants" for describing the underlying information requirements of robot tasks. Using his terminology, he describes the results of Blum and Kozen[14], who address the problem of maze searching theoretically. They show that that a robot can search a 2-D maze using two automata, a single automaton with two pebbles, or an automaton with a single counter (implying that a Turing machine can search a maze using only logarithmic space). O'Kane et al.[74] extend this line of thinking by considering what sensors are necessary for a mobile robot to perform localization by comparing the "information spaces" of different robots.

Dudek et al.[30] develop a taxonomy and classify some hypothetical multi-robot systems and 20 robot collectives from the research literature around the time of their article. They describe the robot collectives using a variety of dimensions including composition, team size, reconfigurability, communication range, cost, topology, and computation capability. Balch and Arkin[7] explore what types of communication are needed for three different multi-robot tasks. Parker et al.[78] and Jones et al.[49] offer approaches for optimally composing a multi-robot system for particular tasks.

2.2.2 Robot Software Architecture

Many robot software architectures have been proposed [3][6][15][38] to give roboticists a set of principles, guidelines, and tools to compose a robot's software system. Most largely ignore the hardware aspects of the system, assuming the hardware necessary for the task is available and the details of the hardware can be abstracted away. Oreback and Christensen[75] review some recent architectures for mobile robots.

Researchers in ubiquitous and pervasive computing are interested in the organization of computer systems that can sense and effect the environment. Approaches like Gaia[89] and Recombinant Computing[31] are "designed to support ad hoc, end user configurations of hardware and software, and provides patterns for data exchange, user control, discovery of new services and devices, and contextual awareness." Qui et al.[83] provide a model

for describing platforms that can be dynamically formed for applications using a publish-subscribe system. Although often these approaches integrate sensors and to some degree support human mobility, they are different than robot systems where autonomy, controlled mobility, and manipulation are key components. Moreover, most of this work is aimed at dynamically forming coalitions of existing platforms for particular applications, rather than as a general guide for design.

2.2.3 Swarm Navigation

Cohen [20] experimented, using simulation, with a technique for collective navigation and mapping by a team of simple mobile robots. The technique uses a similar method to the one presented in this thesis for computing paths in a distributed fashion without any localization or mapping. He compared the paths computed collectively with those computed by A* with complete knowledge of the environment.

Payton et al. [81] present an approach for large scale multi-robot control referred to as "Pheromone Robotics" inspired by biology. They use a system based on virtual pheromones, by which a homogeneous team of mobile robots use short-range communication to accomplish cooperative sensing and navigation. In Payton's work "virtual pheromones" are communicated over an ad hoc network to neighboring robots. In contrast, in our approach information is distributed by the relatively static, embedded, nodes scattered throughout the environment. They used this technique to control a team of 20 mobile robots.

Although not explicitly directed at embedded networks, Parunak et al. developed a technique for coordinating multiple unmanned air vehicles (UAVs) using synthetic pheromones and a multi-agent system [80, 79]. Inspired by pheromone communication in insects, they create potential fields for guiding the UAVs around threats to goal locations in a distributed manner. The technique they developed used uniformly placed (tiled as hexagons) "place" agents to store the pheromone and evaporate it over time, and "walker" agents to spread and react to the pheromone. The walker agents consisted of the UAV agents which physically move over the place agents and "ghost" agents which walk over the place agents virtually. The "place" agents could be implemented in the real world by using some kind of

embedded network. Similarly, Panait and Luke [76] create a system based on pheromones to complete multi-agent foraging in a simulation environment. Interestingly, they explain their approach in a reinforcement learning and Markov decision processes framework. The agents use sample trajectories through the space to find an optimal policy for foraging or navigation. In both of these approaches the environment (i.e. the embedded network) is used as a storage substrate for the agents and only the agents insert and remove information. The network does not "communicate" with itself explicitly.

2.2.4 Embedded Network Navigation

Several robotics researchers have proposed using embedded networks to support mobile robot applications. Both Batalin et al. [10] and Li et al. [57] have developed approaches to navigation using heterogeneous teams composed of mobile nodes and an embedded network. The network of embedded nodes, creates a "Navigation field" [12], which mobile nodes can use to find the their way around. They differ in how they compute this navigation field. Batalin et al. [10] use a sensor network of Motes to aid in coverage, navigation, and task allocation. Specifically, they employ a technique called "Distributed Value Iteration". In their approach, transition probabilities between nodes for each navigation action are determined during deployment. Then, the network uses these probabilities and a cost function to compute a policy of action for the robot. They demonstrated their approach using a Pioneer mobile robot in an office environment using a network of 9 Motes.

Li et al. [57] use a network of 49 Motes to compute a potential field for navigation. This potential field is guaranteed to deliver the mobile node to the goal location via an danger-free path. The field is created by the embedded nodes propagating goal-ness or danger to neighboring nodes. Both Batalin and Li used the Motes hardware platform for their physical experimentation. Alankus et al. [1] developed an approach where a sensor network is used in conjunction with a probabilistic roadmap. They demonstrated their system with a Pioneer mobile robot and 7 Motes.

2.2.5 Embedded Network Coverage

A network of embedded nodes can also aid robots in coverage. Koenig [52] and Wagner [101, 102] devise methods for doing parallel coverage using simple ant robots that communicate indirectly by leaving indicators in the environment. An embedded device can be used as this type of inexpensive indicator with the added advantage that they can communicate with each other. Batalin also uses communication nodes as "markers" in aiding mobile robots in the exploration problem [11]. The embedded nodes offer a suggested un-explored direction for the mobile robots to follow.

2.3 Energy Management

2.3.1 Energy-Management for Mobile Robots

Energy management is a fundamental issue in autonomous robotics. The first autonomous robots, Grey Walter's "Elmer" and "Elsie" [103], had behaviors to search for their recharging station. In much the same way the authors in [94] develop a method for a self-recharging robot to support long-lived operation. The authors of [61] present a power model of a robot based on its mechanics in order to derive the energy costs of different mobile robot coverage patterns. The authors in [25] compare the energy behavior of different communication schemes for multi-robot systems.

There has been recent work to use advanced software mechanisms to support autonomous robot applications. For instance, Player [39] is able to deploy software components in a distributed fashion. The CARMEN[63] software package also relies on an event-based communication mechanism (IPC) allowing flexible deployment of its software components. The MARIE project[23] is concerned with building middleware to connect various robot software systems. They accomplish this by using middleware[92] capable of many advanced software systems techniques.

2.3.2 Software Techniques for Energy Savings

The idea of efficiently deploying computations in a distributed architecture has been proposed before. From a performance standpoint, there has been long standing work in mapping parallel computations onto multiple processors [65]. Taking energy into account, there has been work investigating the use of dynamic voltage and frequency scaling (DVFS) to reduce processor power consumption in distributed real time systems [62]. From an application drive perspective, a solution for efficient distributed speech recognition as been addressed [27], as well as the use of service deployment in wireless hotspots to increase the capabilities of handheld devices [96]. Support for the use of application level adaptations has also been provided by prior work which has quantified possible benefits [21, 34].

2.4 Robots in Education

Robots have been proposed by various educators as a motivating context for learning about many subjects. Papert and his collaborators at BBN originally used wired and wireless physical robots as Logo turtles in the early 1970's, but later relied on a graphical simulation on the screen[33]. Logo was used to teach young children about geometry and problem solving. This inspired Resnick's work[87] which inspired the Lego RCX which is a wildly successful educational robotics platform.

Along with teaching robotics and artificial intelligence[51], the Mindstorms have been used to teach introductory computing. Fagin et al.[32] presented evidence that using robots to teach introductory computing was ineffective. The authors hypothesized the main reason was that access to the robots was limited to the lab and students were unable to work on their assignments at home. In order to overcome this problem, Lauwers et al. [56] follow a similar approach to the one outlined in this thesis and develop an inexpensive personal robot. Their approach is to tether the robot via a USB cable in order to keep costs down and and increase reliability.

CHAPTER III

GNATS: DISTRIBUTION FOR SENSING AND COMPUTATION

We provide evidence that we can leverage some of the benefits of "robot swarms" without the whole swarm being mobile. We use a pervasive network of embedded nodes to aid a simple mobile robot in navigation. The mobile robot does not have the sensing or computational resources available to complete complex navigation tasks alone, but through the use of a large number of even simpler devices pervasively deployed, the robot performs the task. The system together provides a novel data point in the space of robot system architectures. This heterogeneous system of embedded devices and mobile robots puts a natural constraint on the design space of robot systems. The embedded network serves as a pervasive communication, computation, and coordination fabric, while the mobile robots provide task-specific sensing and actuation.

Our system is composed of mobile robots with sensors and actuators supported by an embedded immobile network. We make the following assumptions about the embedded network (which we have implemented in the Gnats platform):

- Limited computation and memory
- Short range communication
- Communication is blocked by navigation obstacles
- Deployed pervasively throughout the environment

We assume the robots and embedded nodes communicate using a short-range medium that is occluded by the same objects that occlude navigation (e.g. walls). Line of sight between nodes implies open space for navigation. The mobile robots in our system are somewhat more capable than the embedded nodes. We assume they support:

- Communication with embedded nodes

- Relative bearing estimation to nearby embedded nodes
- Local obstacle and attractor sensing

Also, we assume that the mobile robots have at least the same communication range as the embedded nodes. The robots need to roughly estimate bearing to the embedded nodes in order to move toward the desired node, but the robots do not need to estimate range. Significantly, there are a few assumptions we do not make. In particular, **we do not assume localization or mapping capabilities** on the part of the robots or the embedded nodes. No mobile robots or embedded nodes are expected to perform localization or mapping. Furthermore **we do not assume the environment is static**. Obstacles to navigation can appear and disappear since we expect the network to automatically adapt to dynamic conditions.

Although having estimates of the map or node positions might improve navigation performance, it is indeed possible for a robot to navigate through complex environments without any map building or localization. The fact that complex behavior such as global navigation can arise in such a simple system makes the approach an interesting candidate of study. Moreover, even compared to map-based navigation, using a pervasive network is particularly beneficial in two situations – dynamic environments and planning for multiple robots. In the first case, the network can detect environmental changes and reconfigure quickly, since a communication message can traverse the network faster than a robot. In the case of multiple robots, instead of the robots building their own maps and somehow merging them, the robots share a common map and common plan. The network could also perform robot coordination, for instance, directing coverage patterns, path de-confliction, or traffic control.

In this work we do not address the deployment of the embedded nodes. We assume they have already been placed in the environment, and sufficiently cover the space. Consider the following scenarios:

- the network is already part of an existing infrastructure–for example, an embedded network in a building, or a sensor network in a forest.

- The network did not exist before the robots began the mission, but the robots have already installed the network as supporting infrastructure.
- A dense sensor network was deployed by aircraft or explosive.

Motivated by these scenarios we believe we can separate the deployment and utilization of embedded networks into two different problems. In this work we only investigate the latter.

3.1 The Gnats Hardware Platform

The Gnats embedded network platform is used to aid mobile robots in navigation. The Gnats platform is flexible and inexpensive, but very resource constrained. The platform is pictured in Figure 1. The devices have 4 IR emitters, 4 IR receivers, a temperature sensor, an input button, and two visible light LEDs, and a PIC16F87 microcontroller for computation. The PIC's oscillator is configurable from software, with a frequency range of 125kHz to 8MHz. It has 4K of programmable flash program memory and 368 bytes of RAM. There is an external 32K serial EEPROM for storage, and an additional 256 bytes of backup EEPROM. A variety of sleep modes result in very long lifetimes and always-on functionality. The sleep modes can be controlled via infrared messages, the button, or by timer. The devices cost approximately $30 to manufacture, permitting large-scale experimentation.

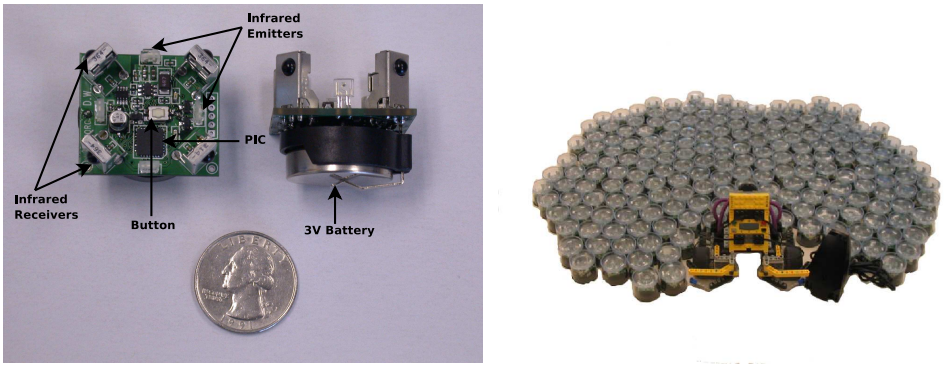


Figure 1: The Gnats embedded network platform.

The output power of the IR emitters is configurable from software. We have measured reliable communication up to 5m in best-case configurations. The Gnats have been

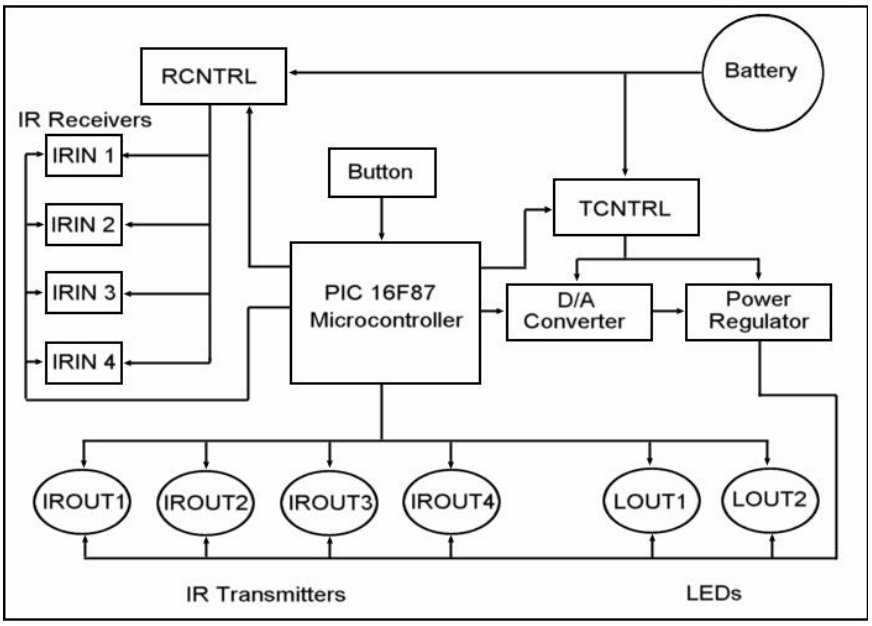


Figure 2: GNAT block diagram.

programmed to communicate with the LEGO Mindstorm/RCX robots allowing them to communicate with both robots as well as the LEGO infrared PC dongle. This enables interaction with a PC, for instance, for downloading new programs, putting the devices to sleep, and reading the EEPROM.

The devices can be programmed individually via the MPLAB integrated development environment through a PICSTART PLUS device programmer. However, the best way to program many devices at once is using the LEGO infrared dongle. The PIC is self-programmable, meaning it can read and write its own program memory while running. This capability enables us to program the Gnats at runtime using the infrared communication system. We have programmed over 100 Gnats in under 3 minutes in this fashion. A system for dynamic code propagation was also developed.

3.1.1 Hardware Description

In our current experimental setup the PIC microcontroller runs at 4MHz. The visible light LEDs are currently used for debugging purposes only, but in the future may be used as a communication point for nearby robots. The IR emitters have a maximum rating of 100mA and with this we have measured reliable communication over ranges of up to 5m in best-case

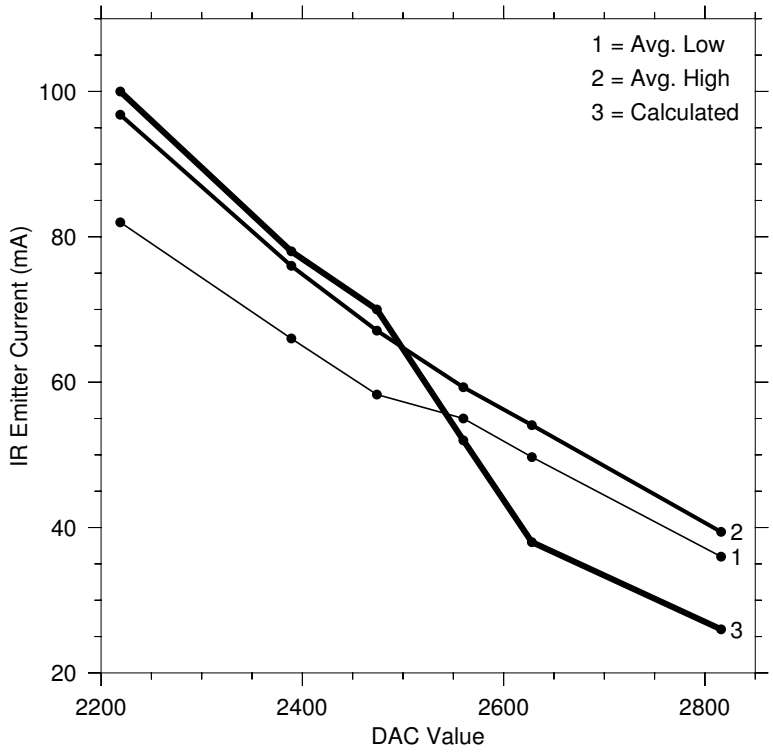


Figure 3: DAC Value vs. IR Emitter Current. Calculated values are based on the emitter data sheet function and simulation. Avg. High and Low are measured values.

configurations.

As outlined above, the GNATs have three sources of input. The first is the button. The button can be configured either to reset the PIC or as a general purpose input for human-GNAT interaction. The second input comes from the infrared receivers. The last input is the programming port. The PIC is programmed via the MPLAB integrated development environment through a PICSTART PLUS device programmer [66]. The programming port can also be used for RS-232 serial communication. Serial communication with a PC improves upon the LED/Button human-interface mentioned earlier. In addition, using serial communication, one of the GNATs can be used as a communication device for a mobile robot. The mobile robot can carry a GNAT onboard to interact with other GNATs embedded throughout the environment.

The block diagram in Fig. 2 shows the basic organization of the hardware. The output power to the visible light LEDs and IR emitters is governed by a power regulator and a D/A converter. By programming the D/A converter via software, we are able to control

the current applied to the emitters. We experimentally determined the range of possible output power values (see Fig. 3).

The entire transmission circuit may be disabled when not in use - such as during sleep times - by setting the TCNTRL (transmitter control) pin with the PIC. Likewise, the receiver circuit may be turned off using RCNTRL (receiver control). Turning the circuits off is a power saving strategy.

3.1.2 Software Description

The following are function prototypes that are defined by the GNATs API. The GNATs software is written in C and compiled by the CCS C compiler for the Microchip PIC microcontroller. (For a more complete description of the GNATs API see [66].)

```
void init(enum CPU_SPEED speed);
void display_led (int8 led_id,
     uint16 dacvalue, uint8 time);
void turn_on_led(int8 led_id, uint16 dacvalue);
void turn_off_led(int8 led_id);
void send_packet_now(uint8 *data,
     uint8 transmitter, uint16 dacvalue,
     uint8 numb_bytes);
int  recv_packet_blocking(uint8 *data,
     uint8 receiver);
```

The `init()` function sets the input/output direction of the peripheral port pins on the PIC microcontroller for transmission and reception of IR data and sets the clock speed to one of eight pre-defined speed constants, ranging from 125 kHz to 8MHz.

`set_output_voltage()` sets the brightness of the visible light LEDs and the IR emitters, `dacvalue` is the value to be programmed into the D/A converter. This value indirectly specifies the brightness (and consequently the power consumption) of the LEDs or IR emitters.

For display and debugging purposes, `display_led()` is used to flash the visible light LEDs. `led_id` specifies either the red or green LED, `dacvalue` is the value to be programmed into the D/A converter before activating the LED, indirectly specifying the brightness (and consequently the power consumption) of the flash, and `time` specifies the duration of the flash in milliseconds. `display_led()` returns after `time` milliseconds have elapsed and the flash has ended.

`turn_on_led()` and `turn_off_led()` allow arbitrary control of the LEDs. However, both LEDs should not be used simultaneously without carefully selecting a `dacvalue` compatible with both of them.

When an application is ready to send a packet of data, it calls the `send_packet_now()` function. `data` points to the buffer to be transmitted. Any combination of `transmitter`s can be used, by OR-ing their IDs together, but the `dacvalue` parameter specifies a common transmission power for all of them. `num_bits` is the number of bits in the packet pointed to by `data`.

`send_packet_now()` returns after the transmission is complete, and does not wait for any acknowledgment from the receiving GNAT.

`recv_packet_blocking()` listens for a packet over IR and stores it in the buffer pointed to by `data`. `receiver` specifies which of the four IR receivers to attempt to receive with. In our current implementation, this function is is called from within an interrupt handler which is triggered by IR activity on any receiver. The interrupt handler determines which receiver(s) is active and calls `recv_packet_blocking()` with the corresponding argument.

3.1.3 Experiments

In the first of three experiments we quantify the maximum distances for reliable communication at various emitter orientations. We used two GNATs for this experiment. One has a fixed orientation and the other's orientation is varied. The node that varies its orientation communicates using only one of its transmitters. We note the distances at which we have reliable communication. The results of this experiment are shown in Fig. 4. The high maximum ranges (5.5 meters) seen are due to the elimination of all fluorescent interference

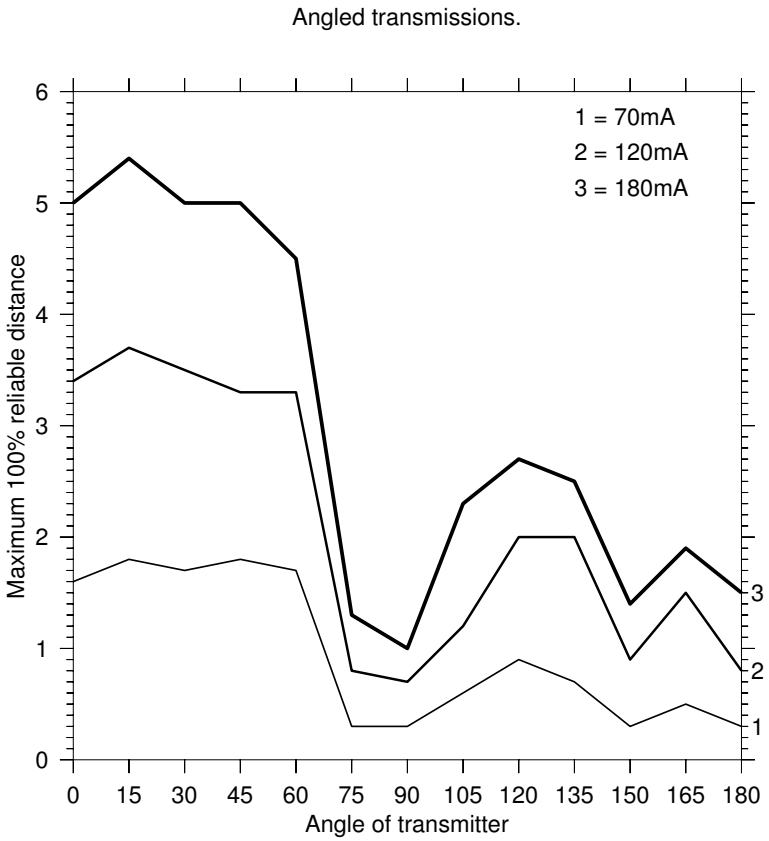


Figure 4: Varying orientation of the sending node using a single emitter. Higher range beyond 90° is most likely due to reflection from aluminum back side of receivers. Values shown for output amperages are estimates based on the emitter data sheet and simulation.

for this experiment. Unshielded fluorescent lighting sends out intermittent bursts of light in a frequency that is detectable by the IR receivers thus interfering with IR communication. Interestingly, as we move beyond 90° we see an irregular range of distances. This is because the rear side of the IR receivers is aluminum and we are getting a varying amount of reflection.

In the next experiment, rather than using only one IR emitter, we use them all. Note that in Fig. 5 when transmitting out of all 4 emitters we have a more evenly distributed, but shorter range.

As an extension of the previous experiments, we wanted to determine how much interference the fluorescent lighting in our lab causes. In this experiment, again we use two nodes. We pick two orientations and vary the distance between the nodes. The results are

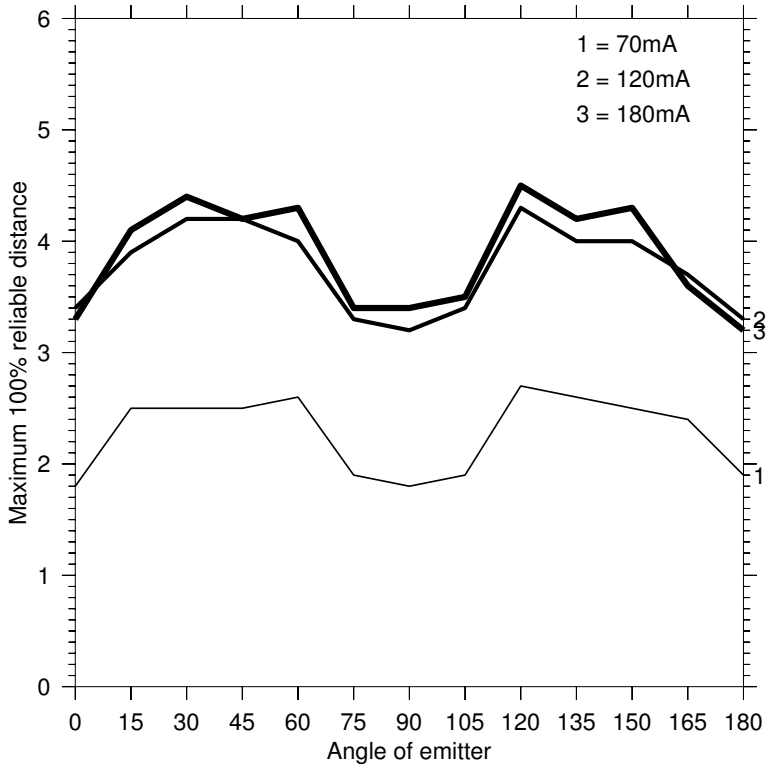
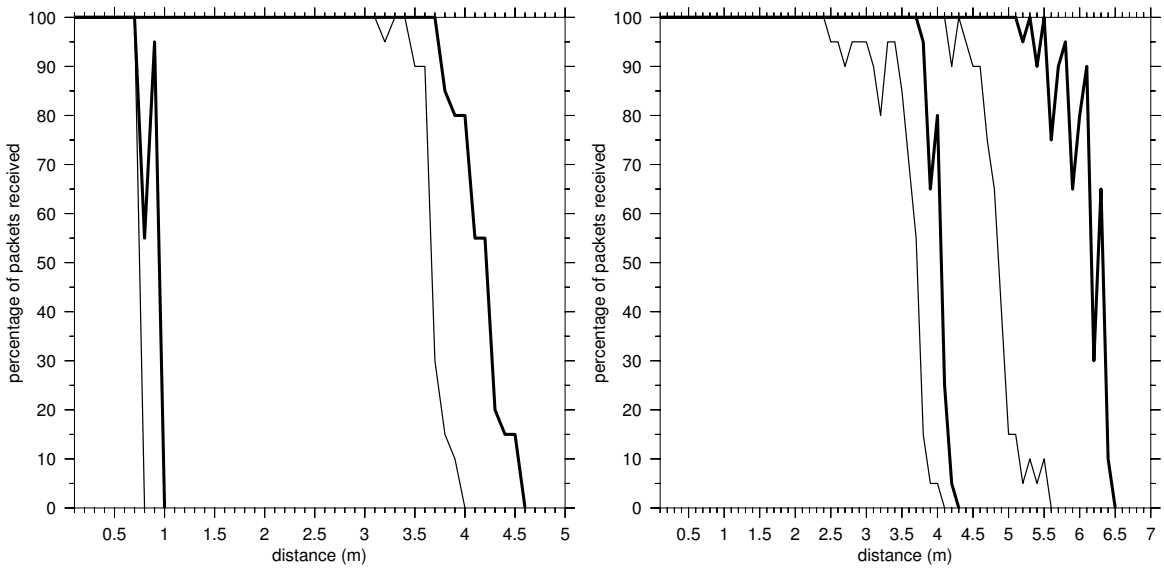


Figure 5: Varying orientation of the sending node using all 4 emitters. Orientations are measured from a single emitter. Values shown for output amperages are estimates based on the emitter data sheet and simulation.



(a) Single emitter.

(b) All 4 emitters.

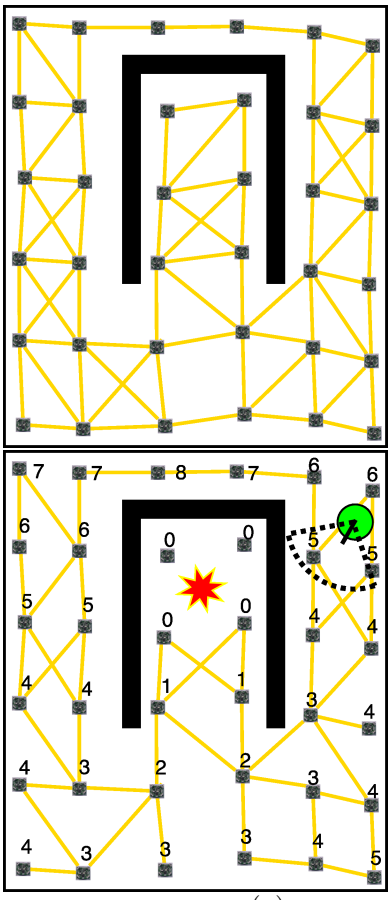
Figure 6: Heavy lines represent no fluorescent lights present. The two rightmost lines are the best orientation for transmission, which in the single emitter instance is 15° and when transmitting on all 4 emitters is 37.5°. The two leftmost lines are the worst angle in all cases: 90°.

summarized in Fig. 6. Note the fluorescent lighting interference can decrease transmission ranges by as much a 1.3 meters.

3.2 The Distributed Path Planning Algorithm

The embedded network can guide, or route, mobile robots in various tasks. We use the network in this work for planning paths to particular goal locations. Either a sensor on an embedded node, or a nearby robot causes one of the nodes in the network to become the goal. In general, several overlay networks can be present simultaneously–for instance, a network for directing coverage patterns may also be present. A dense network could use topology control techniques, or limited mobility, to create a sparse, uniform, overlay for navigation. A mobile robot can then follow the overlay network corresponding to its current task. Figure 7 illustrates the physical path planning algorithm.

Like Payton [81], we assume the communication paths are similar to the navigation paths and use this to propagate navigation information. By using a short-range communication medium that is occluded by obstacles to navigation, the communication paths carve out



(a)

```
structure PathMsg
    int hops
    int sequencenumber
    int gnatid
procedure Init()
    sequencenumber ← 0
    neighbors ←
on event MsgReceived(PathMsg msg)
    Add(neighbors, msg)
    hops ← MaxHops
    for n in neighbors do
        if n.timestamp < ValidTimeout then
            if n.hops + 1 < hops then
                hops ← n.hops + 1
            end if
        else
            Delete(neighbors, n)
        end if
    end for
    if msg.sequencenumber > sequencenumber then
        sequencenumber ← msg.sequencenumber
        TransmitMsg(hops, sequencenumber, gnatid)
    end if
```

(b)

Figure 7: (a) An illustration showing how the network guides a mobile robot to a goal location. (b) The Gnats physical path planning algorithm.

free-space. To create a navigation network a particular goal we use a distributed dynamic programming approach; specifically, we apply the distributed Bellman-Ford algorithm. The Bellman-Ford equation [13] for finding the shortest path from i to j is:

$$D(i,j) = \min_{k \in neighbors} d(i,k) + D(k,j)$$

Where $D(i,j)$ is the path cost from i to j, and $d(i,k)$ is the distance between i and k. It can be used to find the shortest path to a destination from all nodes. The distributed version of Bellman-Ford was created for network routing protocols [35]. In the distributed network routing version, neighbors share their path costs and the distance between nodes is usually measured in hops. We use distributed Bellman-Ford to effectively create a directed graph of shortest paths from every node to the goal. The embedded network can be thought of as "routing" the mobile robots to their destination. However, note that the embedded nodes do not know the global, or local, position of their neighbors, so they are not directing the robot in any direction. Instead, the mobile robot greedily approaches the lowest-valued node currently in its communication range. As it closes in on the node it will come within communication range of that node's parent. The robot continues this until it comes within sensing distance of the goal.

A brief outline of the algorithm follows. First, a device is designated as the goal. Next, the goal node begins to send messages to its neighbors, with its distance to goal (its hop-count) equal to zero, and with increasing sequence numbers. The rest of the network continues to propagate this "goal" information. The Gnats keep a neighbor list with information such as the neighbor's id, the last time the device talked to its neighbor, and the neighbor's hop-count. A neighbor is removed from the neighbor-list if a period of time has passed without any interaction. Then each device looks through their neighbor-list for the neighbor with the minimum hop-count. This minimum hop-count is incremented by one and is propagated as the node's hop-count. The Gnats only propagate information when they receive a message with a higher sequence number than they have seen before. The algorithm is sketched in Figure 7.

3.3 Path Planning in Dynamic Environments

We implemented this system in the TeamBots multi-robot simulation environment. The control systems of the mobile robots were encoded in the Clay behavioral architecture [9] and used motor schemas [5] for the local navigational tasks of moving toward the embedded nodes and avoiding obstacles. In all the experiments we used 2 mobile robots and a group of 16 attractors to represent the goal location. The first robot was placed near the attractors in the center of the environment at the start of the experiment. This robot's purpose was to insert the goal information into the network. This scenario mimics a situation where a robot needs to recruit other robots to a particular location. The second robot was placed in the corner of the environment away from the goal and was responsible for navigating to the goal location. We used 180 embedded nodes. We placed the embedded nodes uniformly across the space, then added error to each node's position by some random amount (error drawn uniformly between 0 and 1 meter). The robots and embedded nodes had limited sensing and communication ranges of 4 meters that were occluded by obstacles. The environment was 36x36m^2.

A sequence of screenshots of a simulation using the distributed path planning algorithm is shown in Figure 8. The screenshots show a tree of shortest paths to the goal. The first screenshot shows the initial plan, and the second shows the plan after a door has closed and the network has reconfigured.

3.4 Multi-Robot Foraging

Given the system of robots and network nodes described above, we would like to solve a multi-robot foraging problem. Foraging is a well-studied, canonical multi-robot task [7, 40]. In this task a robot team is initialized at a "homebase" location, from which they should begin to explore the environment in search of attractor (food) objects. Once a cache of attractor objects is discovered, this information should be disseminated to the other robots, along with a means for them to navigate to the cache. Finally, all of the attractor objects should be collected by the robots and delivered to homebase. We have decomposed the overall problem into the following sub-problems:

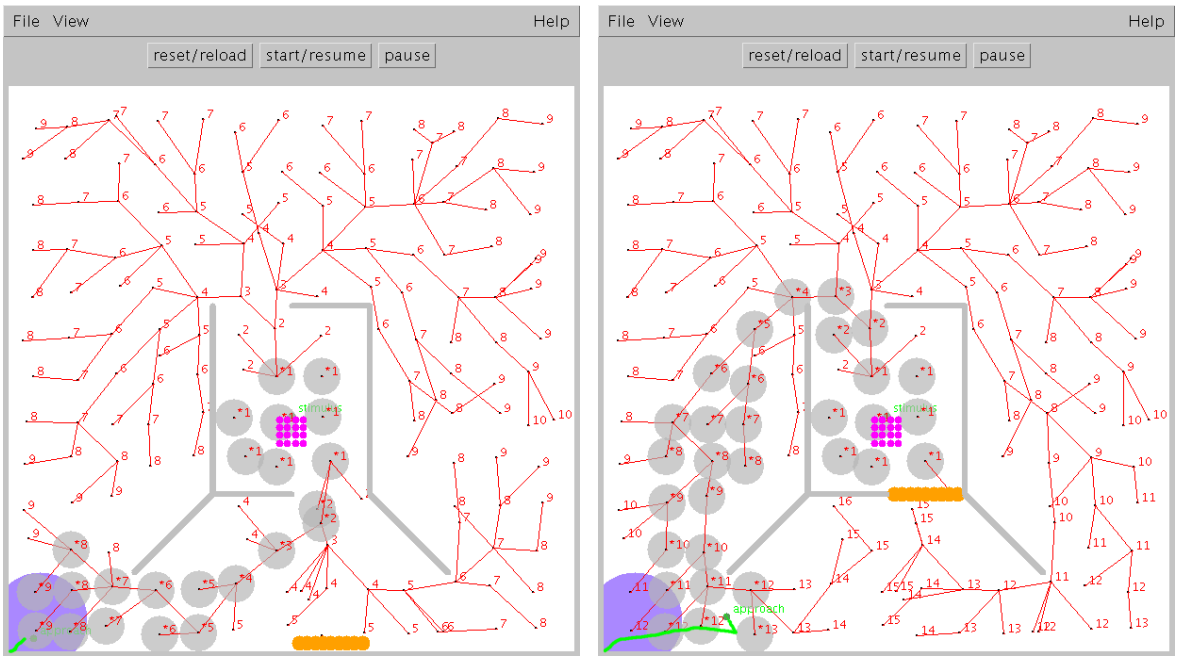


Figure 8: A sequence of screenshots of a simulation using the distributed path planning algorithm. The screenshots show a tree of shortest paths to the goal. The first screenshot shows the initial plan, and the second shows the plan after a door has closed and the network has reconfigured. The numbers indicate the nodes' distances to the goal, the lines between nodes indicate a parent-child relationship. The goal location is represented by the pink circles in the center, the mobile robots by the green circles, their trails by the green lines, and obstacles by the gray and yellow lines.

- Cooperative coverage: enable a team of mobile robots to completely explore the area covered by embedded nodes. For efficiency, robots should avoid traveling through areas already explored by other robots.

- Recruitment: alert the team to new, critical information. In the context of a foraging task, the discovery of a food cache is an example recruitment situation.

- Path planning: Without requiring localization capabilities, provide an efficient route for each robot, located anywhere in the environment, to a goal location.

The embedded network creates *navigation networks* for guiding, or routing, mobile robots in various tasks such as coverage, recruitment, and path planning. We use navigation networks to accomplish three different steps in the task: 1) directing the robots to visit uncovered areas, 2) directing the robots to a discovered goal, and 3) directing them

home. Navigation networks for each of these tasks are present in the network simultaneously. A mobile robot can then follow whichever navigation network corresponding to it's current sub-task goal. We are able to use navigation networks to complete the multi-robot foraging task in complex environments without mapping or localization.

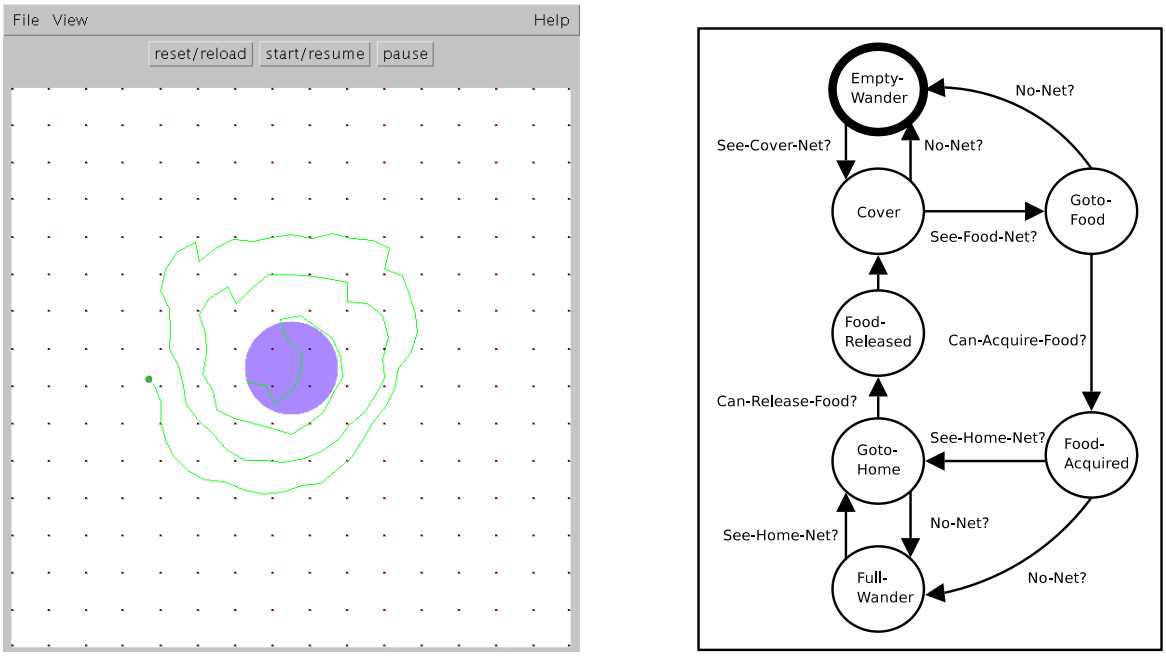
3.4.1 Coverage Navigation Networks

The earlier algorithm for physical path planning can be directly used to direct robots either to the food cache or the nest. For coverage, we considered two mechanisms for building coverage navigation networks. The first is a straightforward application of the goal navigation network, where each unvisited node is a goal. The mobile nodes are then offered paths to reach the closest unvisited nodes. This approach assures all nodes will be visited.

The second coverage navigation network algorithm works in a similar fashion, but orders how unvisited nodes are visited. Our implementation is inspired by Zelinski's [108] approach to coverage. It forces mobile robots to visit the closest nodes first, thus assuring that they find the closest food source to exploit. In this version, if the node hasn't been visited then its coverage value is its value for the home navigation network. This causes the robots to visit all of the nearest nodes first.

In both coverage approaches, if a node has been visited, it uses the default scheme of propagating one of its neighbor's values. However, the second coverage navigation network creates a frontier of embedded nodes that are to be explored next. An example of this is shown in Figure 9. Here we see the mobile robot spiral from its starting position to cover all the closest nodes first.

Although the coverage solution generated is not optimal in the sense of shortest circuit to visit all the nodes (i.e. the traveling salesman problem) it does assure all nodes are visited. Depending on the configuration of the embedded nodes, this can assure systematic coverage of the terrain, even though neither the robots or the embedded nodes have any localization capabilities or a map. In addition, both algorithms can be used in dynamic coverage scenarios by changing their state to unvisited after a certain amount of time has passed. Both algorithms work well for both single and multi-robot exploration.



(a) (b)

Figure 9: (a) An illustration of the second coverage navigation network being followed by one mobile robot. We see the mobile robot spirals from its starting position to cover all the closest nodes first. (b) The behavior state diagram of the mobile robots for the foraging task. The robots start in the empty-wander state.

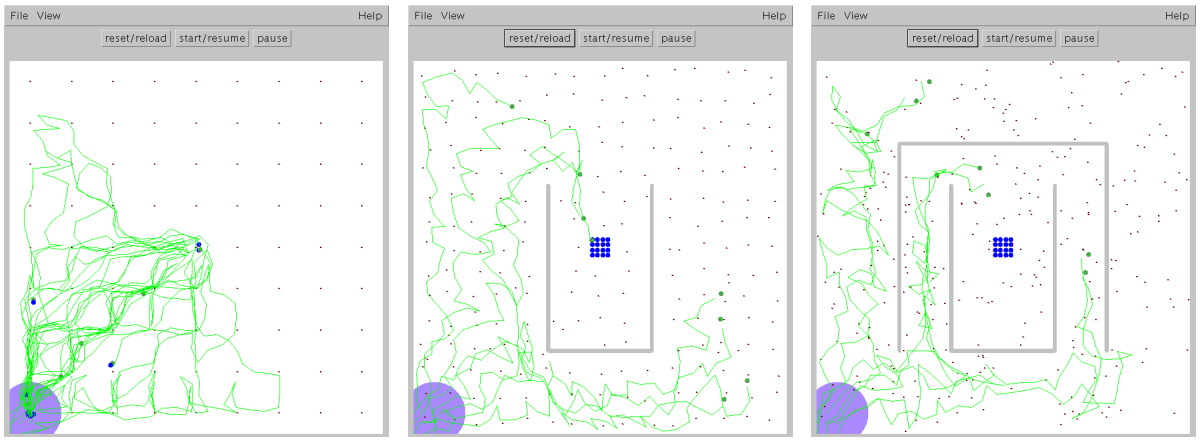
3.4.2 Foraging Simulation Results

We combined the three navigation networks (food source, homebase and coverage) to complete a multi-robot foraging task. We implemented this system in the TeamBots multi-robot simulation environment. The control systems were encoded in the Clay behavioral architecture [9]. The state diagram for the mobile robots which shows when robots switch from one behavioral mode to another is illustrated in Figure 9. In all the experiments we used 8 mobile robots with grippers, and 16 attractors in a group to represent a food source. All the robots had limited sensing and communication ranges of 4 meters that were occluded by obstacles. We tested the technique in three different $36x36m^2$ environments of increasing complexity. The three environments are shown in Figures 10(a), 10(b), and 10(c).

Two key factors impacting how the system performs are the number of embedded nodes and how they are placed. As mentioned previously, when we have a large number of embedded nodes deployed uniformly we effectively have a grid-world and the navigation networks are accomplishing distributed path-planning. Much work has dealt with trying to optimally deploy a sensor network for these tasks. In this research, however, we assume that the network is approximately uniformly distributed, but with random placement error. Placement error in a real system could be due to error in deployment or changes over time. Since our approach does not depend on the embedded nodes being localized, it is robust to changes in placement.

We ran experiments with 81, 121, 169, 225, 289, and 361 embedded nodes. Additionally, we varied the error in placement using the following technique. First, we placed the nodes uniformly across the space, then added error to each node's position by some random amount, the average distance from original position was varied: from 0, .5, and 2, to 10 meters. In the case of 10m average error, placement is essentially uniform random. Each experimental configuration was run 10 times. The graphs show mean performance, with errorbars denoting standard deviations.

The first set of results show the average time, in timesteps, to cover 95% of each environment using the frontier navigation network. Cover times of 7200 timesteps (the simulation timeout) were used for experiments that did not cover 95% of the area. The results of

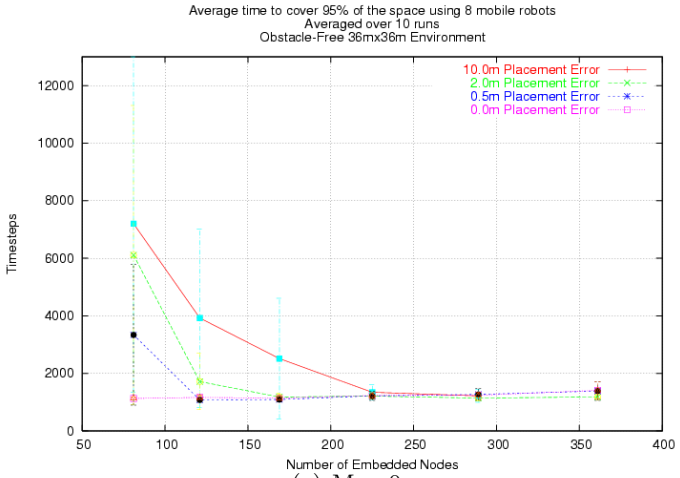


(a) Map 0 (b) Map 1 (c) Map 2

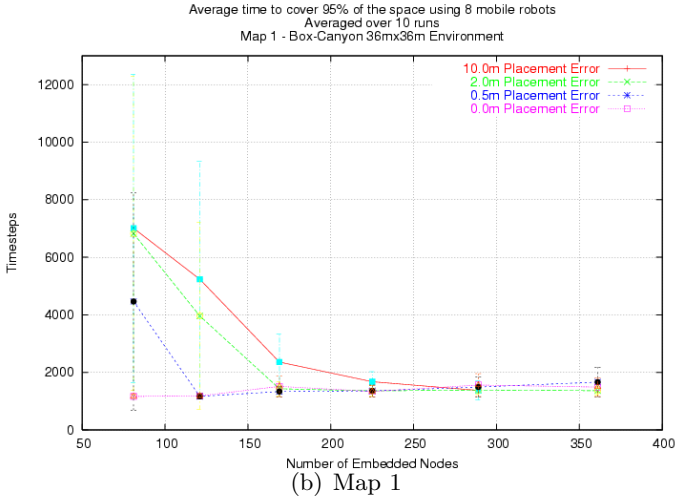
Figure 10: The simulation environments. The blue circles in the center represent the food source, the blue circle in the bottom left corner is the robots' starting position, and the green lines are robots' trails. (a) Map 0 is the first obstacle-free environment. A sample configuration is shown with 81 embedded nodes distributed uniformly throughout the environment without any error in placement. (b) Map 1 is a more complicated environment with a box canyon. A sample configuration is shown with 225 embedded nodes distributed uniformly with .5 m of error in placement. (c) Map 2 the most complicated environment with two box canyons. A sample configuration is shown with 289 embedded nodes distributed uniformly with 10m of error in placement.

the first obstacle free environment are shown in Figure 11(a). We see that as we increase the error in placement, more nodes are needed to maintain the same level of performance. This is due to the fact that with a small number of nodes and large amount of error, the navigation network is disconnected and isn't able to guide the robots in covering the area. Instead, they must rely on a random walk to cover the space. The results of the second and third more complex environments are shown in Figures 11(b) and 11(c). These results show us that more embedded nodes are needed to overcome the error in placement as the complexity of the environment increases. In addition, the coverage times are independent of the complexity of the environment once enough embedded nodes are used to establish connectivity.

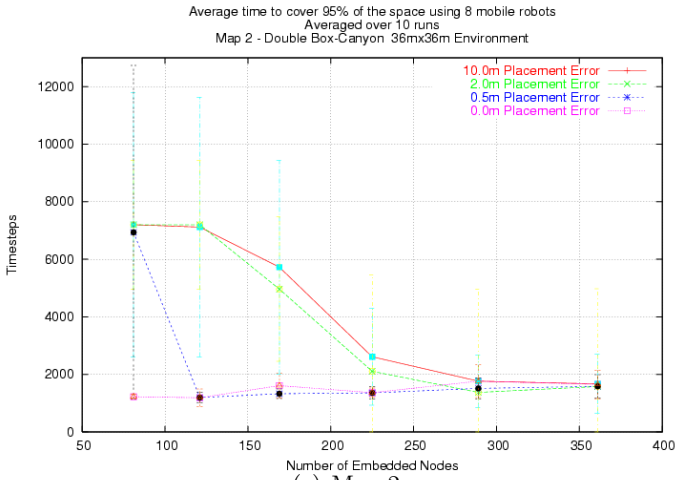
The second set of results show the average time between retrieval and delivery of each attractor in the foraging task. This value correlates with the efficiency of the computed path. These numbers do not include any exploration time or time spent returning to the food source, but only the delivery portion of the foraging task. The results of the first



(a) Map 0



(b) Map 1



(c) Map 2

Figure 11: The average time to cover 95% of the area as a function of the number of embedded nodes, error in their placement, and the environment.

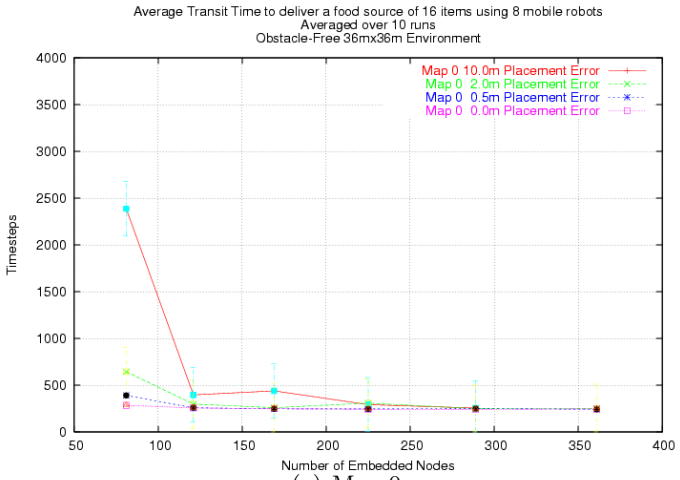
obstacle free environment are shown in Figure 12(a). We see that with uniformly arranged nodes, the number of nodes does not increase performance. But as with the coverage task, as we increase the error in placement, more nodes are needed to maintain the same level of performance. The results of the second and third more complex environments are shown in Figures 12(b) and 12(c). Unlike the coverage task, as the complexity of the environment increases from Map 0 to Map 1, it takes longer for the robots to deliver the attractors. This is sensible since the shortest path available to the robots is longer. The path length for Map 2 is not longer than Map 1, so we do not see any additional time for delivery in this environment.

The last set of results are for the complete foraging task. They show the average time to deliver each attractor. Delivery times of 7200 timesteps (simulation timeout) were used for undelivered attractors. The results shown in Figures 13(a), 13(b), and 13(c). These numbers reflect the results we saw in the two subtasks of foraging previously. The critical factor in all the experiments in the performance of the navigation networks was connectivity.

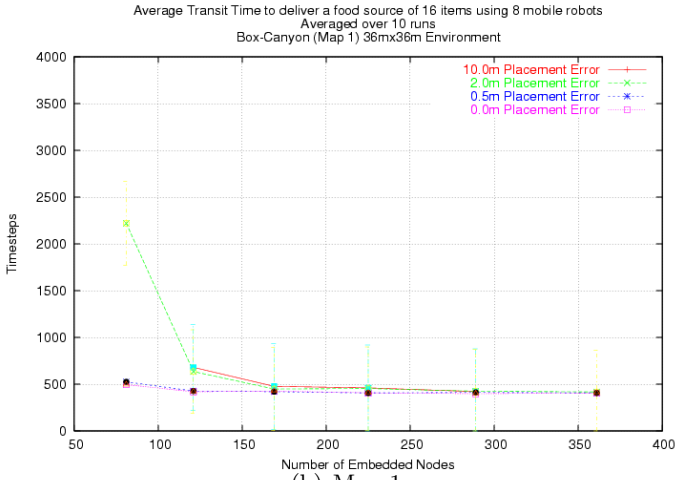
3.5 Experimental Results

The Bellman-Ford algorithm was implemented successfully on the Gnats hardware platform. We chose two sets of experiments to conduct. In the first, we try to quantify how close the Gnats paths are to the optimal paths. Second, we give experimental evidence for how a simple mobile robot can use the paths to navigate a maze.

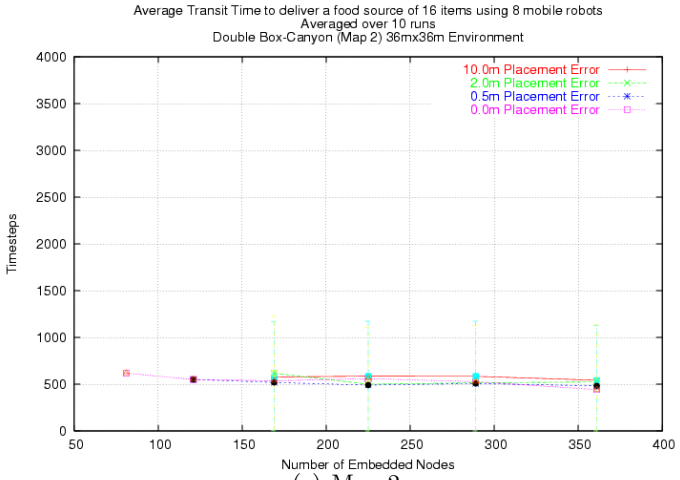
Typically, propagation of the goal information takes on the order of a couple of minutes in networks of around 100 nodes. Of course, this time is a function of environmental complexity, network deployment, ambient interference, etc. Precise measurement of propagation time in our experiments was prohibited by the lack of accurate clocks on the devices. During the experiments the devices log their local connectivity information (neighbor-list) and their distance to the goal (hop-count) periodically. After the experiment, we uploaded the devices' EEPROMs to a PC for post-processing. Using the data we create a graphic describing the experiment (e.g. experiments using 60 Gnats are visualized in Figure 14).



(a) Map 0

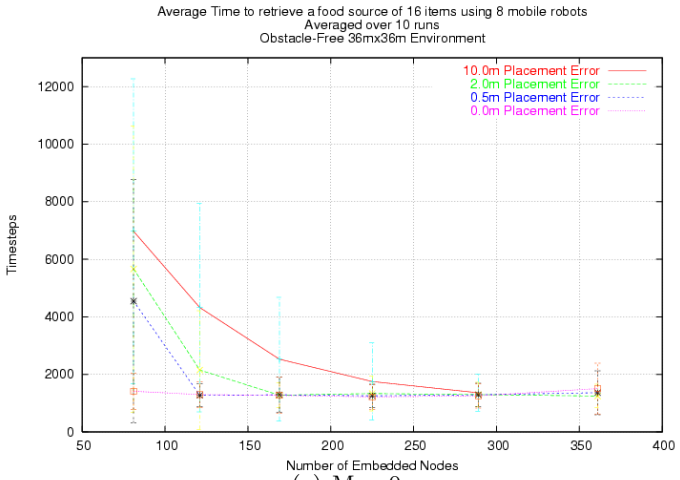


(b) Map 1

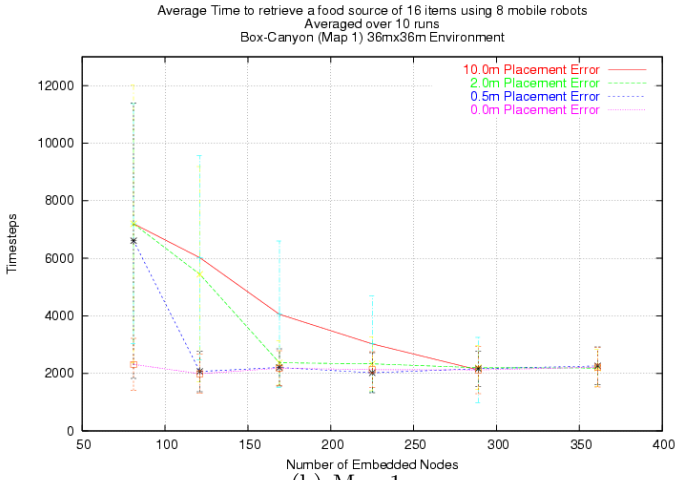


(c) Map 2

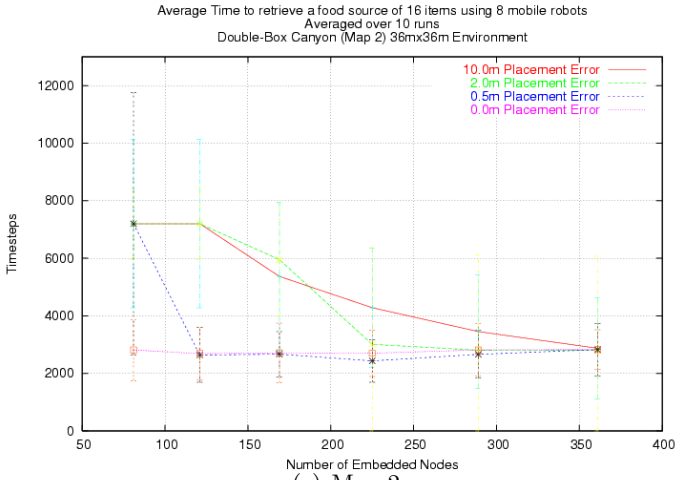
Figure 12: The average time between retrieval and delivery of each attractor as a function of the number of embedded nodes, the error in their placement, and the environment.



(a) Map 0



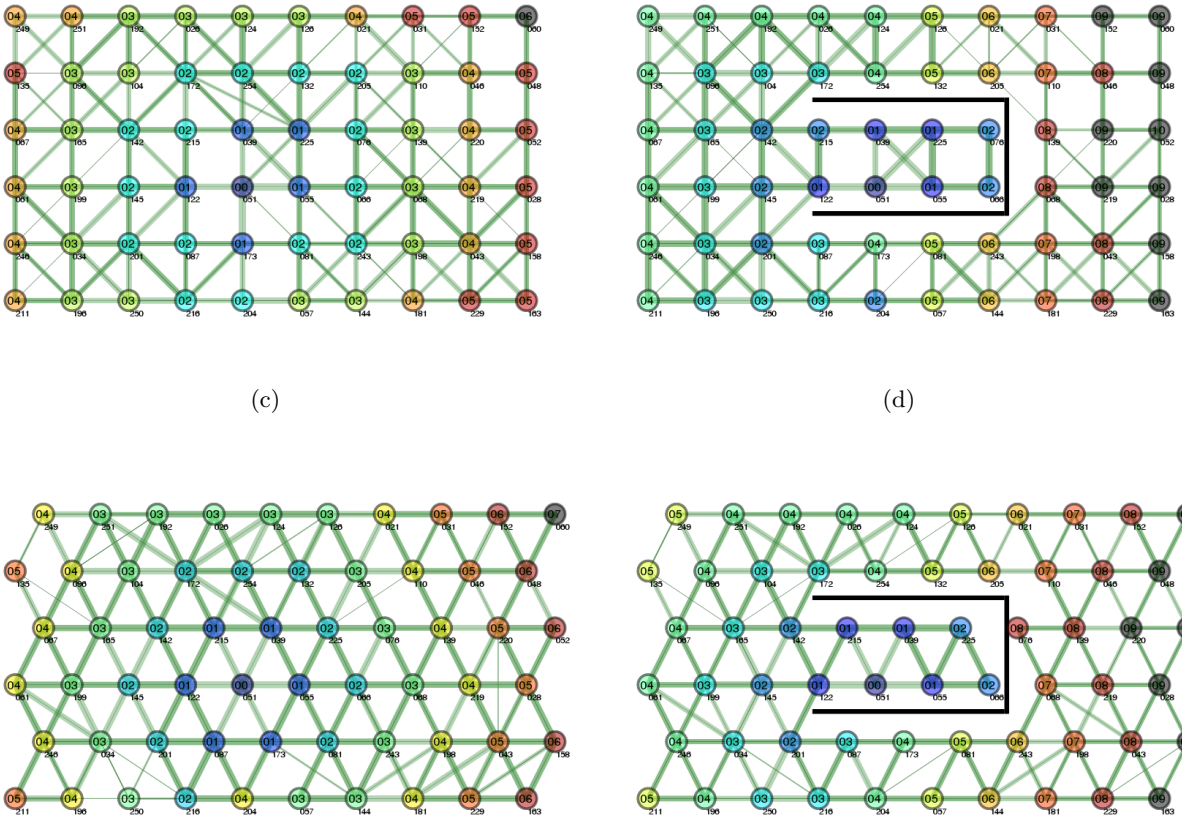
(b) Map 1



(c) Map 2

Figure 13: The average time to deliver each attractor as a function of the number of embedded nodes, the error in their placement, and the environment.

(a) (b)



(c) (d)

(e) (f)

Figure 14: (a) The rectangular arrangement with no obstacles. (b) The hexagonal arrangement with a box canyon. (c-f) Visualizations of the data from a network of 60 Gnats showing the hop-counts and connectivity when node 51 acts as the goal node. Thicker lines represent more connectivity.

Table 1: The average path lengths (mean/stdev in ft.) of the optimal path, the expected Gnats path, and the actual Gnats path.

	Rectangular		Hexagonal	
	No Obstacles	Box-Canyon	No Obstacles	Box-Canyon
Optimal Path Length	8.62	9.40	8.56	9.29
Expected Gnats Path Length	10.01	11.70	9.46	10.74
Actual Gnats Path Length	11.98	12.8 6	9.87	12.29
Average Difference from Optimal	1.71/4.66	3.52/9.51	1.33/1.12	3.06/8.76
Percentage Longer than Optimal	**18.60/49.96**	**34.35/85.78**	**13.89/10.50**	**28.62/73.46**

3.5.1 Quality of Distributed Path Planning

In motivating our technique, we assumed the communication graph approximates the path-planning graph, but how good is this approximation? After all, the Gnats infrared communication is affected by ambient infra-red interference, unexpected interference by nearby nodes, reflections, hardware failure, etc. In order to answer this question, we empirically analyze two types of deployments, rectangular and hexagonal arrangements, in two different environments. Although these arrangements seem somewhat ideal and unrealistic, topology control techniques[106] can be used to create similar uniform network overlays for navigation from real (i.e. non-uniform) sensor network deployments. Also, if the embedded nodes are mobile, these types of patterns can be created with the techniques offered in [46, 60].

In all configurations, the environment is covered by a connected network of 60 Gnats. The first environment has no obstacles and the second environment has a box-canyon obstacle. In all cases the nominal communication range is set to 2-3 ft. In the rectangular configuration, the Gnats are separated by 2 ft, and the hexagonal arrangement is the rectangular with every other row shifted. The environments are pictured in Figure 14.

For each configuration we ran 10 different experiments. For each experiment, a different node was designated as the goal and we logged the communication connectivity and hop-counts to EEPROM. Graphics depicting the network connectivity and hop-counts for four particular experiments are shown in Figure 14. The data collection in one experiment (rectangular-obstacle-254) failed so we do not include that in the analysis.

To better understand the quality of the computed paths, we compare three different path lengths. First, we compute *Optimal Path Length*, the length of the direct path a robot

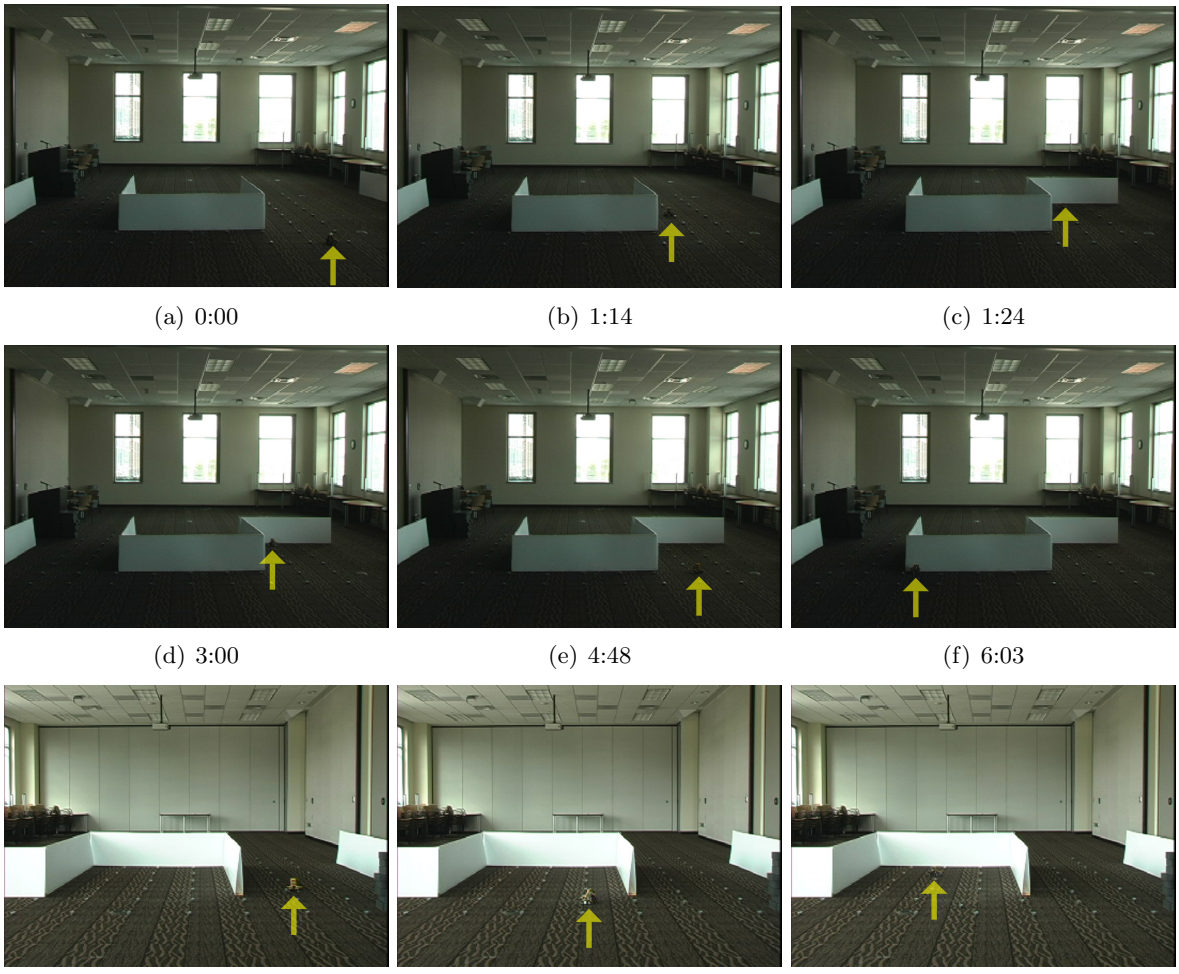
could take if it used a map of the environment instead of the Gnats. Second, *Expected Gnats Path Length*, the length of the path along the waypoints prescribed by the Gnats, assuming a perfect 2.5 ft. communication range. Finally, *Actual Gnats Path Length*, the length of the actual path in light of imperfect or unexpected communication between Gnats during our experiments.

The last two path-lengths model a robot moving from node to node. In other words, the *Actual (Expected) Gnats Path Length* is the length of the path a robot would use to navigate through a field of Gnats. To compute the path-length in cases where a node has multiple parents, we average the parent's path-lengths. At any node the robot moves to one of the node's parents with with equal probability. Thus, the path-length for a node is the average of the quantity - the distance to each parent plus the parent's path-length.

For all 40 experiments we compute the optimal path-length from each node to the goal. Also, we compute actual Gnats path-length, i.e., if a robot were to go from node to node following the computed hop-counts as described above. Using these two numbers we calculate, on average, how much longer the Gnats path is than the optimal path. The results are presented in Table 1. We see that the hexagonal deployment is on average, closer to the optimal than the rectangular. More importantly, the computed path-lengths are on average (over all configurations), only 24% longer than optimal.

3.5.2 Mobile Robot Results

In addition to propagating messages from the goal, the network of Gnats are programmed to interact with a LEGO Mindstorm/RCX robot. The mobile robot is chiefly composed of a LEGO base, two motors, two bump sensors, a Hitachi H8 processor with 32K of RAM, and an infrared transceiver. The base of the robot is constructed such that Gnats can pass through the middle of its body, or they will be pushed to its side. The mobile robot was programmed with the BrickOS software environment allowing us to modify the communication protocol stack to be compatible with the Gnats. The Gnats and the RCX robots exchange arbitrary messages over a 2400 baud infrared communication channel at ranges up to 3ft.



(a) 0:00 (b) 1:14 (c) 1:24

(d) 3:00 (e) 4:48 (f) 6:03

(g) 6:35 (h) 7:58 (i) 14:44

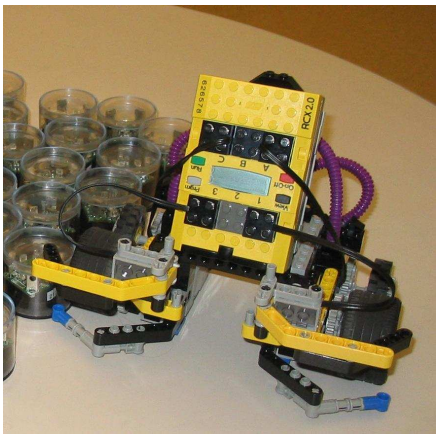
Figure 15: Photos of the robot (arrow) making its way through a dynamic maze. (a-b) The robot makes it way to the goal location in the center of the box canyon. (c) A new obstacle (a wall) is introduced. (d) The network reconfigures and offers the robot a new path. (e-f) The robot begins driving down the new path. (g-i) The robot finishes the maze (images shown from the other side of the maze).

As described above, the Gnats communicate with each other to compute their distances to the goal. Then, the robot uses a very simple algorithm to navigate through the environment. The robot continually polls nearby Gnats (approximately 3 times a second) for their hop-count. When the robot has not talked to any Gnats (either initially or after a time-out), the robot will spin in a complete circle keeping track of the lowest hop-count it sees. The robot then drives forward as long as it hears its current destination node or another node of lower hop-count. Note, once the robot has chosen a destination node, it will not consider other nodes of the same hop-count until it times out, thus preventing oscillations between nodes of the same hop-count. If the robot loses contact with its destination node, it turns until it's realigned with the current destination node or another node of lower hop-count. The robot will time out completely after some long idle period. This method of spinning and constant communication provides the robot very rough azimuth information about nearby Gnats.

The global navigation information provided by the Gnats is complemented by a simple reactive response to its bump sensors. If the robot bumps into something, it backs up and turns a small amount away from the collision. This method allows the robot to traverse small obstacles not accounted for by the Gnats. The global path planning provided by the Gnats combined by the robot's local reactive navigation results in the robot navigating through complicated environments. The robot was able to use a network of 156 Gnats (see Fig. 17) to navigate through a complex maze in under 10 minutes. Fig. 17 shows the path of the robot, automatically generated from a video using background subtraction.

Additionally, when the environment changes, the Gnats are able to detect this, replan, and offer the robot a new path. The speed of this response is a function of the timeout values of both the Gnats and the robot. Given a very reliable communication channel these timeouts values can be arbitrarily small. Sadly, our communication channel is not so reliable and we must have modest values for these timeouts. Despite the latency in reconfiguration (on the order of minutes), the time spent reconfiguring is much faster than having the robot perceive, map, and re-plan.

Results from an experiment in a dynamic environment are shown in Fig. 15. In this



(a)

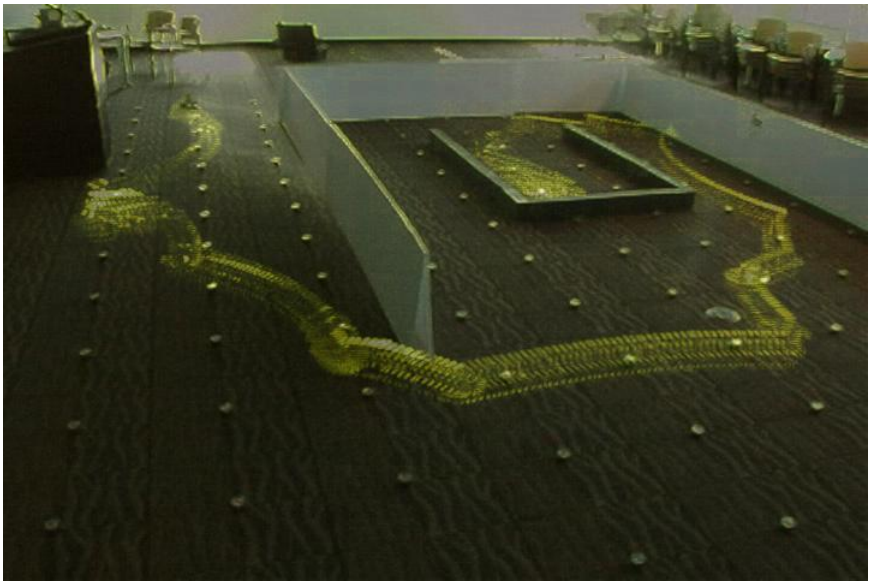
```
procedure Init()
  hops ← MaxHops
  gnatid ← 0
  SetTimer(MaxTimeout)
on event MsgReceived(PathMsg msg)
  if msg.hops < hops or msg.gnatid = gnatid then
    SetTimer(0)
    hops ← msg.hops
    gnatid ← msg.gnatid
  end if
procedure Main()
  Init()
  loop
    if BumpLeft() then
      BackupAndTurnRight()
    else if BumpRight() then
      BackupAndTurnLeft()
    else if GetTimer() > LargeTimeout then
      Init()
      Spin360Degrees()
    else if GetTimer() > SmallTimeout then
      ScanTurn()
    else
      GoStraight()
    end if
    TransmitRequestMsg(gnatid, hops)
  end loop
```

(b)

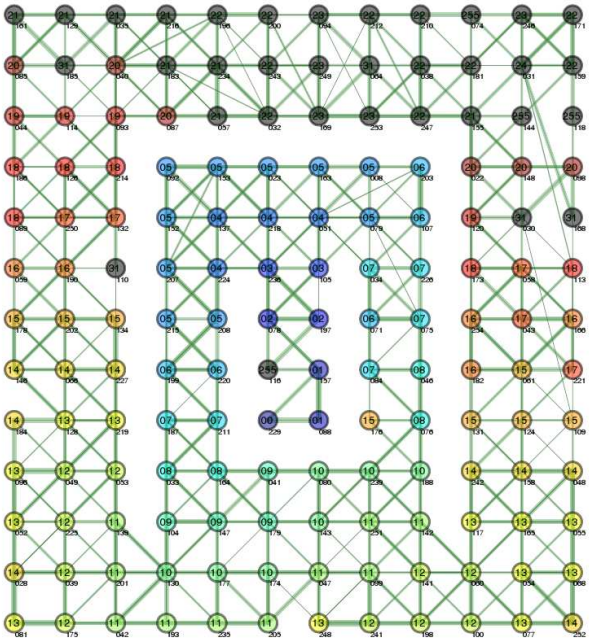
Figure 16: (a) A picture of the mobile robot used in our study. (b) The mobile robot's algorithm for navigating through the Gnats.

scenario a new obstacle is introduced into a box-canyon environment approximately 1.5 minutes into the experiment. The network of 60 gnats reconfigures in less than two minutes, presenting the robot with a new path to the goal. The robot reaches very close to the goal in under 8 minutes, but due to communication trouble, is not able to reach the final goal position until the 14:44 minute mark.

Because of the mobile robot's simple navigation algorithm, and even simpler hardware, its trajectory through the network is not as straight as we would hope. The robot occasionally loses communication with its destination node causing it to turn in circles until it regains contact. The robot performed approximately 35 full spins in the experiment in Fig. 17. The hardware mismatch between the infrared communication systems of the robot and the Gnats is the chief cause of the unreliability.



(a)



(b)

Figure 17: (a) Two views of the maze. (b) The communication graph of 156 Gnats when node 229 is assigned to be the goal. The path of the robot is drawn in black.

3.6 Reflection on Architecture

The Gnats system shows that it is possible to use pervasive sensing, memory, and communication resources to significantly reduce the resources needed by mobile platforms.

For example, consider the navigation task. The pervasive nodes act as a distributed memory, relieving the robot of having to store a map of the environment, and thus reducing its memory requirements. In addition, the nodes also provide distributed computing services, actually computing the navigation path on behalf of the robot. By using pervasiveness, not only do we reduce the resources needed by the mobile platform, but we can also increase the performance. For example, by shifting the sensing resources to the pervasive infrastructure, the environmental changes can be reacted to very quickly. Rather than a robot having to come within sensing distance of an environmental change, the event can be detected by the pervasive infrastructure. Detecting changes in the environment is highly temporally task, but not spatially local, thus it is advantageous to make the pervasive network responsible for this part of the navigation task. In the case of multi-robot foraging, the pervasive network acts as a collective memory concerning not only path information, but also including coverage information. In this respect, the pervasive nodes act as a distributed memory and communication and coordination service for the mobile robots.

CHAPTER IV

AUTOPOWER: DISTRIBUTION FOR ENERGY

Autonomous robot systems have to manage their energy wisely in order to complete their missions. Typical approaches seek to limit the energy usage of each robots system platform, using general or application-specific power management methods. Examples of the latter are energy-efficient motion or sensor planning. We seek to avoid solutions that limit individual robots' sensing or actuation capabilities in order to ensure longer lifetimes for their computing subsystems. Toward this end, we present a distributed computer systems approach to runtime power management. Specifically, we present the AutoPower model for characterizing and manipulating the software systems that execute on robot platforms.

There are multiple reasons why AutoPower adjusts robot software systems' energy behavior without considering tradeoffs that may be realized by changing robots' sensing or motion behaviors. First, solutions like AutoPower can support efficient energy usage without requiring roboticists to change the behaviors they are seeking to develop or utilize. AutoPower's energy management solutions, therefore, are a general service to any robot system regardless of its particular application, architecture, robot platform, etc. Additional energy savings attained via application-specific solutions will simply enhance AutoPower-based solutions. Second, as autonomy increases in robot systems, energy usage for computation will constitute a substantial portion of total energy consumption (i.e., battery drain). Third, as we look to teams of robots, system energy consumption is increasingly impacted by communications across team members.

Decisions about which software components to run on the computer systems of distributed robots depend both on current application needs and on available system resources. Application-level decisions that affect the use of available system resources include the following:

- **Which components should run?** Concrete examples are which localization, path-planning, or vision routines to execute on robots.

- **How should the components run?** Application needs dictate how many particles to use for Monte Carlo localization or what grid granularity to employ for D*.

- **When should the components run?** Real-time constraints are determined by current application context or mission parameters, determining for instance, the periodicity and deadline for the obstacle-avoidance routine.

The AutoPower approach seeks to retain for applications the ability to run the components they desire and in the fashion suitable for their current needs. Rather than limiting the application's ability to use system resources, the approach supports application needs by seeking to automate how such resources are used:

1. **Where should the components run?** For example, should we 'offload' localization computations to a remote computer?

2. **How should the components interact?** Should robots communicate using 802.11a or should they use bluetooth communications? Should component middleware use pull- vs. push-based communications?

Stated more explicitly, AutoPower attempts to answer the following question: **How can we deploy a robot software system to maximize the lifetime of the system while still meeting application QoS requirements?** In addition, AutoPower looks to manage energy at a system-wide level, rather than maximizing individual systems (e.g. communication protocols, operating system scheduling) of individual robots (e.g. voltage scaling, sensing frequency).

The contributions of the AutoPower approach, then, are as follows. First, the approach provides a method for modeling the energy behavior of robot software systems, based on energy profiles of the software components' computation and communication behavior. Second, along with energy models of robots' computing platforms, these profiles are then used

to estimate a particular software deployment's energy lifetime. Third, this estimated lifetime function is used as an objective function to statically map software components to maximize the lifetime of the robot team's system platforms. This deployment is optimal as long as the components follow their nominal energy behavior. Fourth, we highlight the need for dynamic reconfiguration. If component behaviors change, perhaps due to changes in mission profile, or if there are infrastructural changes like variations in connectivity, the system should be reconfigured dynamically.

4.1 The AutoPower Energy Model

Our first goal is to model a robot software system's energy behavior. The creation of such a model requires three main components: (1) energy characteristics of computational platforms, (2) energy characteristics of communication mediums, and (3) execution profiles of software components.

The first key component is the energy characteristics of the computational resources. It is well known that the precise power consumption of a workload can vary with respect to use of the processor, much less the additional consumption of resources like memory, external buses, etc. While fine grain power models attempt to capture all such usages, the AutoPower approach focuses on the scalability of power modeling, by offering a first order approximation of total system power usage. The goal is to capture key system-level tradeoffs. AutoPower attains this goal by summarizing the energy costs of computational resources in terms of joules per millisecond of computation. This allows us to describe the energy demands of our software simply by latency measurements, rather than direct energy measurements, which are more time consuming and costly to gather.

In this work the computational model is developed based upon real hardware. Specifically, the Intel Sitsang experimental platform is used, shown in Figure 18, as a typical on-board computational platform for robots. The Sitsang is based on the Intel PXA255 XScale processor and is aimed at embedded platforms like PDAs and robots. This platform would be ideal for applications such as robots since the PXA255 provides integrated solutions for peripherals such as flash, memory, infrared, USB, and others. Moreover, the

Figure 18: The Sitsang embedded Linux platform.

solution strikes a solid balance between performance and power. Though the XScale core lacks a floating point unit, it can be clocked up to 400 MHz, and is accompanied with 64 MB of RAM, 32 MB of Flash memory, and supports dynamic voltage and frequency scaling. The platform itself is capable of various wireless communications such as bluetooth and 802.11a via expandable slots.

To develop our computational energy model, various workloads are executed on the Sitsang platform, and platform power measurements are performed using a Tektronix TDS5104B oscilloscope, Tektronix TCP202 current probes, and Tektronix P6139A voltage probes. It should be noted that though the Sitsang is equipped with a LCD display, since these might not be attached on a deployed robot, the LCD, backlight, and framebuffer DMA were disabled for our measurements. The workloads utilized were various media based encoders/decoders that are typical of systems such as robotics. When running at 400Mhz, the average computational overhead when idle is 1.34W, and 2.27W when active.

The second component of the system energy model involves describing the communication infrastructure. The communication resources are characterized by the average amount of energy (joules) needed to communicate one bit of information. 802.11a, Bluetooth, and GSM (mobile phone network) are considered as possible communication mediums. The

	802.11a	Bluetooth	GSM
Bandwidth (kbps)	54000	1500	116
Sleep Power (W)	0.010	0.000036	0.000165
Send Power (W)	1.498	0.095	1.815
Send μjoules/bit	0.028	0.064	15.647
Receive Power (W)	1.22	0.095	0.165
Receive μjoules/bit	0.022	0.064	1.422

Figure 19: Energy characteristics of communication resources.

specific energy characteristics of each radio are taken from [37] and are reproduced in Figure 19.

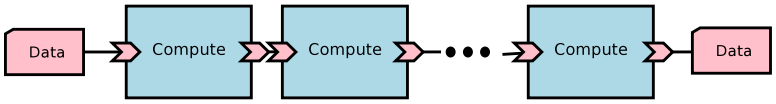


Figure 20: Application Chain: Our application execution model.

The final piece of the energy model is concerned with the software components that must actually execute in the system. In order to derive energy costs, software components are modeled as chains of computation and communication. Each node in the chain does some amount of computation and passes along data that serves as input to the next computational node. The nodes are described in the applications chains by their average computational latencies, and the links of the application chains are described by the sizes of data communicated between two adjacent nodes. The model is illustrated in Figure 20. In order to accommodate application-level requirements (formulated as quality of service (QoS) constraints) into the model, each chain must specify its timeliness requirements (e.g. rate) as well as any hardware constraints like the need for the presence of certain sensor devices or human operators. The overall modeling process is visualized in Figure 21.

4.2 Application Scenario

In order to demonstrate our energy modeling approach, a search-and-rescue mission is used as a guiding application throughout this paper. Specifically, the scenario considered is where a multi-robot team is tasked with searching a building for victims. The lifetime of the system is defined as the amount of time until one robot depletes its energy reserves.

The robot team consists of three mobile robots, all with laser-scanners and odometers.

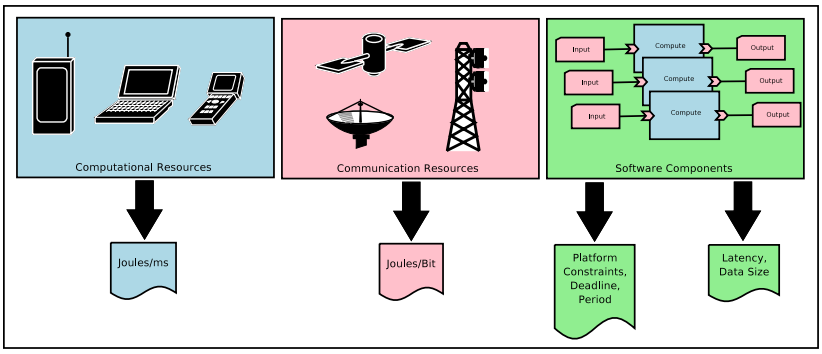


Figure 21: The process of building an energy model.

Two of the three robots are equipped with cameras. The third robot acts primarily as a communication relay. The robots communicate to a base-station (a laptop) operated by a human operator. Also, the robots can use the base-station computer for off-loading computation if necessary. The base-station can communicate with the robots using either GSM or 802.11a (if range permits) communication. The robots have embedded Linux computers on-board and can communicate to each other using 802.11a or bluetooth (again, if range permits). Lastly, the computers are also able to route messages between each other using a shortest-path routing algorithm.

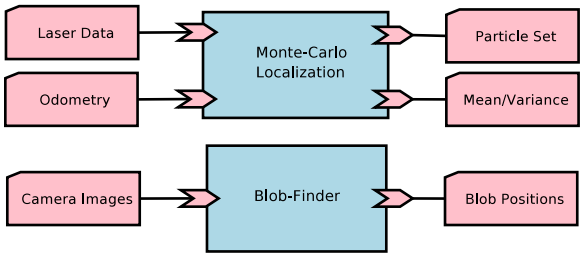


Figure 22: Two application chains for the search-and-rescue mission.

We focus on two software components that will be executed in support of this mission. First, a piece of vision software that detects possible victims. Our particular implementation (blob-finder) locates blobs of different colors in an image. The information from the blob-finder is communicated to the human operator for victim/non-victim classification. The application demands the blob-finder run 4 times a second.

In addition, the robots use Monte-Carlo Localization to estimate their position in the building. The robots may need to report their positions to the human operator, but not

necessarily continually. The localization components should execute once every two seconds. The two application chains for these components, as described above, are illustrated in Figure 22.

4.3 Two Software Execution Profiles

Given the application described in the previous section, energy profiles are created of Monte-Carlo Localization (CARMEN[63]) and Blob Finding (CMVision[17]). While relevant to our driving application, these applications are also representative of typical robots software. First, both implementations are taken from popular open-source robot software distributions. Second, the two pieces of software have very different computational behaviors, which makes them an interesting pair to energy-profile. Specifically, MCL makes heavy use of floating point operations, whereas blob-finding primarily uses integer operations. This distinction causes them to differ substantially with respect to their characterization by the AutoPower energy model. The Sitsang platform lacks a floating point unit resulting in a bigger performance gap between the laptop and Sitsang when running localization compared to running the blob finder.

Table 2: Energy profiles for the two software components.

	Localization	Blob-Finder
Period (sec)	1.0	.25
Laptop Latency (sec)	0.0035	0.01048
Sitsang Latency (sec)	0.85	0.115
Input Size (bytes)	1059	230415
Output Size (bytes)	805	200

The average latencies and data sizes of the two applications are profiled on the computational platforms (Intel laptop, Sitsang) and recorded. The localization module is run using a simulation of two different environments. The robot in the simulation navigates from a start location to a goal position 10 times. The average localization latency is noted. The blob-finder is run on three different data-sets collected using a Logitech web-cam. The blob-finder runs on each data-set 50 times, and the average latency for processing a frame was collected. The results are shown in Table 2. Note, the large disparity in computational performance between the SitSang and the laptop.

4.4 Methodology and Results

4.4.1 System Lifetime Analysis

As an initial step towards analyzing system-level tradeoffs with robot software systems, we built a tool to compute the expected lifetime of our search-and-rescue team given our energy models, the initial energy of each robot, a description of the network (including link types and connectivity), and a particular allocation scheme. In order to limit the design space, a shortest path routing scheme is assumed for all of our network topologies.

In order to calculate system lifetime, first the base power cost of each robot's on-board computer is determined. This is done by taking the base idle power consumption (1.34W), and including the sleep power of every radio that is needed for network links. Then the application chains are mapped across the system as specified by the allocation scheme. The period information of an application chain is used along with the radio characterizations to assess the communication costs to robots. Similarly, periodicity and computation time provide computational energy costs.

To validate that all application requirements are met, the utilizations of the radio and computers are computed. If any of these values exceed 100%, the allocation scheme is deemed impossible. Otherwise, "lifetime" of the system is calculated as a function of the maximum average power consumption of all the robots in the system.

4.4.2 Static Allocation Results

Using the lifetime analysis tool, given the search-and-rescue mission previously described, the space of possible allocations and networks is exhaustively explored. Specifically, all three robots perform localization (`Loc`), and Robots 2 and 3 have cameras attached and require blob finding (`Blob`) on images. One particular scenario is illustrated in Figure 23. In this instance, the two robots with cameras have off-loaded their blob-finding component to the base-station. The robots are fully connected with bluetooth, and the relay-robot is communicating to the base-station using 802.11a communications. In this scenario the relay-robot routes messages between the base-station and the robots with the cameras.

In Table 3, the results for some interesting configurations are reported. The table in the

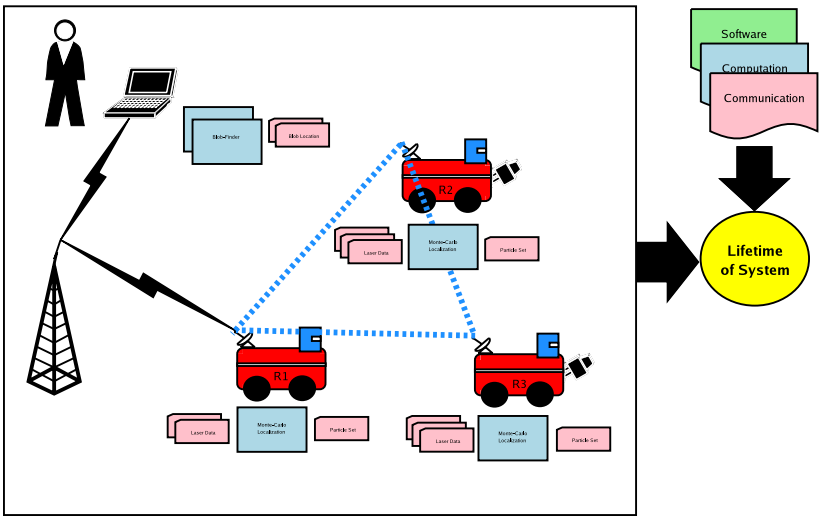


Figure 23: An example scenario.

figure provides the lifetimes for various network configurations and component allocations. The "baseline" configuration is when all components execute locally and all network links use 802.11a-based communications.

The maximum lifetime configuration occurs when the network is fully connected with 802.11a and the robots offload all computation to the laptop. The lifetime of this configuration is approximately 57% better than the case where the robots perform all computations locally on board. This is an interesting result because, from Figure 19, it would seem that for power reduction, the robots should use the bluetooth radio which consumes less power. However, utilizing the more efficient radio prevents certain offloading, and therefore has a negative impact on the entire system. This type of counterintuitive tradeoff highlights the importance of the system-level deployment mechanisms offered by the AutoPower approach.

Also interesting are situations where it is unrealistic that the entire team is connected by 802.11, for instance, due to range constraints. Consider the case where the relay robot is connected to the base-station by GSM and the robots are connected by bluetooth. The relay robot's communication link is the bottle-neck in this situation. The expected lifetime of the base-line (i.e., everything executed locally) configuration is 64% of the maximum lifetime configuration. We also observe that the most efficient allocation for this network configuration is to have the robots offload their localization to the laptop. This allocation

comes closer to the optimal expected lifetime, 78% of the maximum expected lifetime.

Finally, we investigate a scenario in which robots begin the mission with uneven energy reserves. In this situation, the relay robot (Robot-1) has half the energy of the other two robots. Again, interesting configurations are tabulated in Table 4. In this scenario, the optimal configurations are less obvious. When the system is fully connected, the optimal configuration is for the robots to communicate with the base-station using GSM, and for the robots to communicate with each other using bluetooth. The robots "shift" their localization responsibilities away from the robot with the least energy. When Robot-1 is the only robot connected to the base-station the optimal configuration is also interesting. Robot-3 runs both blob-finders and Robot-2 runs the localization for itself and Robot-3. Robot-1's localization is off-loaded to the base-station.

4.4.3 The Need for Dynamic Allocation Support

Thus far, our results have illustrated the benefits of utilizing an optimal allocation scheme versus a basic local execution policy for robot software systems. To realistically utilize these software offloading schemes, however, there must exist system support for dynamic allocation. To support this statement, consider two types of changes that may trigger a dynamic re-allocation. The first type is infrastructural changes like those due to changes in network connectivity or link characteristics. The second type is due to changes in application behavior. That is, the needs of an application, or the chain descriptions in our model, may change as scenarios unfold.

Revisiting our optimal scenario, where all nodes are interconnected by 802.11a and all computations are offloaded, let us assume that Robot-3 experiences a fault with its hardware, and must utilize GSM to contact the laptop, and bluetooth for inter-robot communications. Due to utilization constraints alone, the existing allocation scheme cannot continue while providing the required QoS. Reverting back to a local computation scheme provides only 80% of the lifetime compared to the optimal deployment, where `Loc-1`, `Loc-2`, and `Blob-2` are performed by the laptop, `Loc-3` is offloaded onto Robot-1, and `Blob-3` is performed locally on Robot-3.

Table 3: Expected lifetimes when all robots start with 64,000 joules of energy.

Lifetime (sec)	Normalized	`Loc-1`	`Loc-2`	`Loc-3`	`Blob-2`	`Blob-3`	Laptop-Robot Net	Robot-Robot Net
FULLY CONNECTED								
45715.2	**1.57**	**laptop**	**laptop**	**laptop**	**laptop**	**laptop**	**802.11a**	**802.11a**
29451.2	1.01	r1	r2	r3	r2	r3	802.11a	802.11a
32976.0	1.13	r1	laptop	laptop	r2	r3	GSM	802.11a
29121.6	1	r1	r2	r3	r2	r3	GSM	802.11a
45714.4	1.57	laptop	laptop	laptop	laptop	laptop	802.11a	Bluetooth
29450.4	1.01	r1	r2	r3	r2	r3	802.11a	Bluetooth
33146.4	1.14	r1	laptop	laptop	r2	r3	GSM	Bluetooth
29254.4	1.00	r1	r2	r3	r2	r3	GSM	Bluetooth
ONE RELAY ROBOT								
41820.0	**1.44**	**laptop**	**laptop**	**laptop**	**laptop**	**laptop**	**802.11a**	**802.11a**
29451.2	1.01	r1	r2	r3	r2	r3	802.11a	802.11a
35262.4	1.21	laptop	laptop	laptop	r2	r3	GSM	802.11a
29451.2	1.01	r1	r2	r3	r2	r3	GSM	802.11a
36181.6	1.24	r1	laptop	laptop	r2	r3	802.11a	Bluetooth
29586.4	1.02	r1	r2	r3	r2	r3	802.11a	Bluetooth
35432.0	1.22	laptop	laptop	laptop	r2	r3	GSM	Bluetooth
29586.4	1.02	r1	r2	r3	r2	r3	GSM	Bluetooth

Table 4: Expected lifetimes when robot-1 starts with 32,000 joules and the other two robots start with 64,000 joules of energy.

Lifetime (sec)	Normalized	`Loc-1`	`Loc-2`	`Loc-3`	`Blob-2`	`Blob-3`	Laptop-Robot Net	Robot-Robot Net
				Fully Connected				
23697.6	1.33	r2	r2	r3	r3	laptop	802.11a	802.11a
18335.2	1.03	r1	r2	r3	r2	r3	802.11a	802.11a
23694.4	1.33	r2	r2	laptop	r3	r3	GSM	802.11a
18333.6	1.03	r1	r2	r3	r2	r3	GSM	802.11a
23696.8	1.33	laptop	r2	r2	laptop	r3	802.11a	Bluetooth
18335.2	1.03	r1	r2	r3	r2	r3	802.11a	Bluetooth
23860.8	**1.34**	**r2**	**r3**	**laptop**	**r2**	**r3**	**GSM**	**Bluetooth**
18439.2	1.03	r1	r2	r3	r2	r3	GSM	Bluetooth
				One Relay Robot				
23694.4	**1.33**	**laptop**	**r2**	**r2**	**r3**	**r3**	**802.11a**	**802.11a**
18333.6	1.03	r1	r2	r3	r2	r3	802.11a	802.11a
20812.8	1.17	laptop	r2	r2	r3	r3	GSM	802.11a
17833.6	1.00	r1	r2	r3	r2	r3	GSM	802.11a
23692.0	1.33	laptop	r2	r3	r2	r3	802.11a	Bluetooth
18332.0	1.03	r1	r2	r3	r2	r3	802.11a	Bluetooth
20947.2	1.17	laptop	r2	r3	r2	r3	GSM	Bluetooth
17932.0	1.01	r1	r2	r3	r2	r3	GSM	Bluetooth

Next, consider application triggered re-allocations. For instance, once the mission is over, i.e., all the victims were successfully rescued, the robots may not need to use their cameras any more. In other words, during the robots' return only localization may need to be run. Previously, when the relay robot is providing the communication link back to the base-station, the configuration using 802.11A is the optimal configuration concerning total expected lifetime because 802.11 has to be used to off-load the blob-finders. When blob-finders no longer need to run, there is more flexibility and Bluetooth is able to support off-loading the localization computations.

Finally, with support for dynamic re-allocation, the lifetime of the team could be extended even when there are no infrastructural or application changes. Previously we considered the best possible static allocation. But, by considering re-allocations, it is possible to extend the lifetime of the team even more by dynamically moving components. In other words, at a given decision point, an important system characteristic is the relative energy capacity of each robot. Given an allocation that does not consume energy evenly across nodes, then, it is possible that even without changes to application chains or infrastructure, a run-time re-allocation may further improve lifetime beyond a single deployment strategy. The ability to extend optimal lifetimes with this type of reallocation will depend upon the efficiency and architecture of the system-level support for dynamic re-allocation.

4.5 Reflection on Architecture

The AutoPower system uses a heterogeneous system of robots to overcome the energy limitations of parts of the system. By making use of pervasive computing resources with unlimited energy budgets, the budgeted parts of the system can offload the energy expensive computing tasks.

CHAPTER V

IPRE: DISTRIBUTION FOR EASE-OF-USE AND COST

Teaching introductory computing within a context has proven to be highly effective[36]. Robots provide a real, concrete, physical context for learning about computing. Robots give computer programs embodiment. The repetition of a looping construct is physically actualized by a machine performing repetitive actions. Of course, a computer is already a physical object, but a very complex, opaque physical object. Introductory students can't see the interworkings and dynamics of a computer processor, but by programming a relatively simple robot, students work with a more observable, concrete computational artifact[100]. They work with a computational artifact they can see, touch, and *completely* wrap their head around. Students can use their bodies to design and debug their programs; in the language of Papert[77], they can "play turtle." Moreover, rather than displaying "hello world" on a computer window, a physical LED is illuminated or a tone is beeped, giving more credence to the idea that computer can affect the real world. A message increasingly important in attracting students to computing as a discipline.

We have leveraged distributed robot system design in order to create a personal robot for computing education. The distributed robot system uses a standard laptop for computation as well as providing additional input and output modalities, such as a joystick, text-to-speech, and on-screen graphics. The laptop commands a mobile robot over a bluetooth wireless link. The mobile robot has on-board computation, sensing, and actuation. This particular allocation of resources was decided in order to satisfy our pedagogical goals, thus we looked to maximize robustness, ease-of-use and expressiveness, while minimizing cost.

The IPRE robot system has been used by thousands of students across the country and internationally. Although initially the system allowed students to interact with the Scribbler robot using the Python programming language in service of introductory computing, other users have implemented bindings in a variety of languages including C++, C, Java, Matlab,

and Scheme, for a variety of robots including the iRobot Create in support of a variety of courses including artificial intelligence, robotics, and computer architecture.

5.1 Design Goals

The primary, and defining, objective of our particular robot system for computer science education was that the robot should be personal. Each student should have their own personal robot. This was motivated by prior research that had shown shared, expensive robots that were limited to in-lab use were ineffective for learning computer science[32]. When compared to traditional methods of instruction, students had less access to the technology and their performance suffered. Moreover, it was hypothesized that a robot that could be personalized and act as a source of expression would be engaging. This seemingly simple objective of a personal robot, substantially shaped the design of the robot system in terms of cost, size, reliability, and ease-of-use.

Table 5: An overview of the objectives and design decisions of the IPRE robot system.

Primary Objective	**Secondary Objectives**	**Primary Design Decisions**
A Personal Robot for CS Education	Cost	Wireless Tethering
	Ease-of-use	No Assembly Required
	Size	Fits in a Lunch Box
	Reliability	Open Architecture
	Personalization	

5.1.1 Making Robots Personal

There are a variety of consequences of the personal robots objective. Some are more obvious than others. The fact that each student has their own robot means the robots have to be inexpensive. Our target price for the robot was that of a college textbook. This secondary objective of low-cost is a direct consequence of the personal robots primary objective. Other consequences are less obvious. A less obvious consequence of each student owning their own robots is that *every student has their own robot* – that is a lot of robots! This fact greatly amplifies the roles of reliability and ease-of-use in terms of making the logistics of the class feasible. An unreliable or user-unfriendly robot can be problematic for a student, but thirty unreliable or hard-to-use robots would be an absolute nightmare for an instructor.

The large number of robots also greatly impacts the size of the robot and use of wireless communication, since many students will be competing for shared space (in multi-robot systems speak, physical interference) and bandwidth. Although you can shrink the robot to a size amenable to a desktop, and potentially tether the robot via a cable, this limits the robot's mobility.

5.1.2 Wireless Tethering

Wirelessly tethering the robot to a laptop or desktop computer was a fundamental design decision. The primary advantage of tethering the robot is that the student can interact with the robot through a standard, familiar computer interface. Consider most introductory computing programming assignments. Let's break the activity into three steps: composing, debugging, analyzing. Students write their programs, debug them, and then analyze or demonstrate how well they work, for example, collect some performance information or pass some series of test-cases.

Approaches that don't rely on tethering the robot typically rely on the computer only for composing the program and rely on the robot's (often limited) input/output for the last two thirds of the activity. By tethering the robot for the entire activity, students can use the computer in both the debugging and analysis stages as well as in the composition. This mode of operation, coupled with an interactive dynamic programming environment (e.g. Python) offer the students a more responsive interface for interacting with the robot than a compile-download-run approach.

A secondary advantage of tethering the robot to the student's computer is the added resources the computer offers. Not only in terms of processing and more user-friendly interaction, e.g. the familiar keyboard/mouse and computer monitor, but also the computer's Internet connection, support for USB peripherals like joysticks, and support for sound. By tethering the robot we can compensate for the robot's deficiencies. Due to cost constraints, the robot won't be able to do certain things. For example, the Scribbler robot isn't able to perform text-to-speech even though most standard computers can easily perform text-to-speech. Using this fact, we can make the robot system talk to the students, even if the

actual mobile robot lacks this capability. Many students swear the robot is talking to them, when in reality the sound is coming from their laptop's speakers.

By tethering the robot to a computer (i.e. creating a distributed robot system), we construct a more general, capable robot.

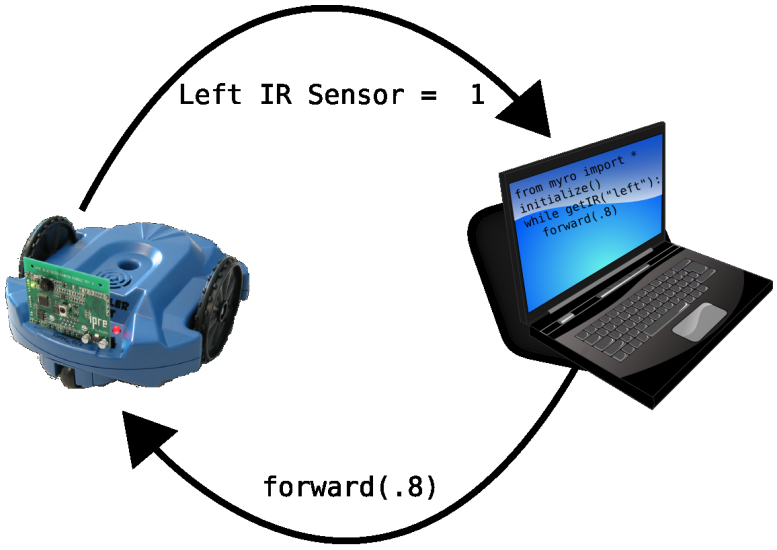


Figure 24: An illustration of the tethered "robot as peripheral" approach.

A disadvantage of tethering the robot is that it adds another component to the mix. A component that adds expense and can fail. Most college students have access to a computer already, and many schools require laptops, so the decreased reliability is the primary disadvantage. In fact, in classes that use our robot system, problems are often caused by strange PC problems rather than robot faults.

A second disadvantage of wireless tethering is that wireless communication can be a source of trouble. Connectivity trouble can surface in terms of establishing the initial connection and in supporting many users communicating with their robots at one time in one place. The use of Bluetooth helps with the latter, but the former is still a bit tricky. Bluetooth handles multiple students (25 students in a class is commonplace) quite well, but the initial connection procedure can be a little confusing for users.

The advantage of wirelessly tethering the robot is that without a wire the robot is free to roam around the lecture hall or student's dorm room. This opens up the possibility of a mobile robot that can leave the student's desk. This is a major win in terms of engagement.

Having the robot tethered via a cord can be beneficial in terms of communication reliability and offering power, but limits the motion of the robot.

5.1.3 No-Assembly Required

Another important design decision that was made was to limit or completely eliminate any assembly or soldering. Principally, the robot system was targeted at computer science education, so effort spent on the mechanical or electrical engineering activities was to be minimized. Again, when considering the logistics of a class where there are as many robots as students, skipping assembly and more importantly re-assembly, makes the classes more feasible.

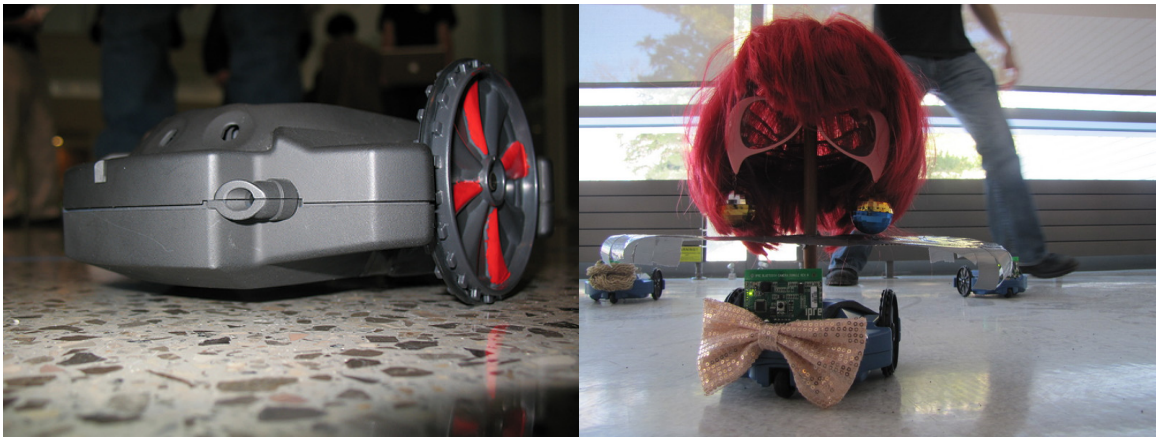


Figure 25: Examples of Scribbler customization by students.

In some sense, providing a ready-to-go, standardized robot makes the robot *less personal*. However, just because assembly is not required, does not mean it should be totally abandoned, for example, decorating the robots can be a great way to personalize the robot. A key lesson learned is that although assembly should not be required, allowing opportunities for expandablity can help make the robots more personal (see Figure 25).

5.1.4 Open Architecture

The robot system should be highly accessible (i.e. from a large number of operating systems and programming languages), expandable and interoperable. Also, communication should follow a common protocol and avoid proprietary solutions. Along with the wireless tether

model, we adopted a client-server architecture allowing educators to use whatever programming language they desire. Moreover, we used common standards for communication such as RS-232 and the Bluetooth Serial Port Profile (SPP). This allowed easy integration with all common operating systems and programming languages. No additional operating system drivers needed to be developed.

5.2 System Architecture

Of the many potential robot platforms for teaching introductory computing: the Lego Mindstorms, the Parallax Scribbler and Boe Bot, the iRobot Create, and more expensive research robots like the Pioneer, none satisfied all our design criteria. Therefore, we created a brand new platform for computing education.

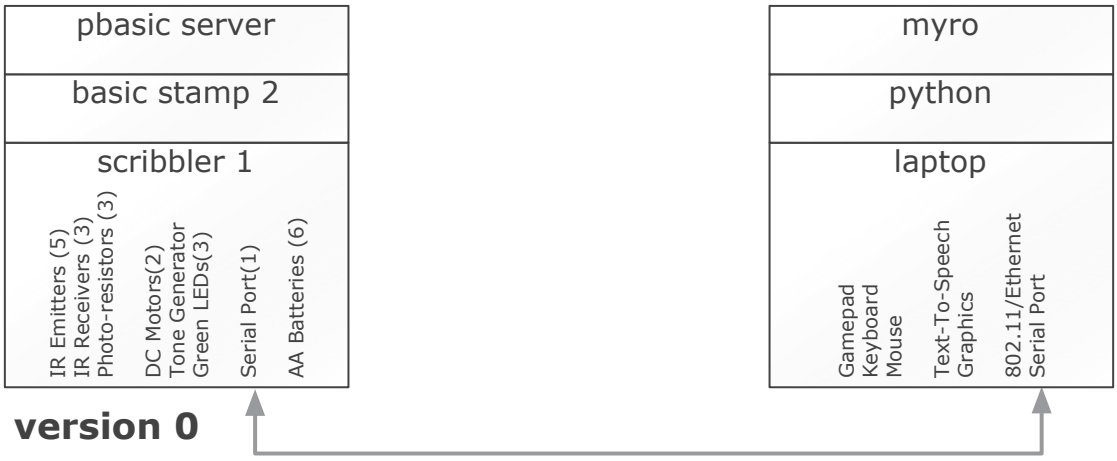


Figure 26: Version 0 of the IPRE robot system architecture

The system evolved over four iterations as seen in Figure 41. The system architecture follows a layered, synchronous client-server model. At the base of the robot system is the Scribbler robot designed and manufactured by Parallax. The Scribbler robot was primarily designed for young children. Parallax provides a visual programming environment which translates the drag-and-drop programs into PBASIC. Those programs are then compiled into bytecode and downloaded to the Scribbler's BasicStamp microcontroller. We chose the Scribbler as our robot base because it was inexpensive ($80) and required no assembly, but could be expanded via its serial port.

The Scribbler robot is powered by 6 AA batteries, has two infrared obstacle detectors (a single receiver, and two emitters), three photo-resistor light sensors, two infrared line detectors, a stall motor-current sensor, a reset button, three green LEDs, a speaker, and two DC motors. The scribbler has a serial port used for programming its Basic Stamp microcontroller, but this port can also be used for communication. Notably, this version of the scribbler does not have wheel encoders or rechargeable batteries.

```
from myro import *

init()

thresh = 50
while timeRemaining(60):
   l = getLight('left')
   R = getLight('right')

   # decide how to act based on sensors values
   if L - R > thresh:
      # left is seeing less light than right so turn right
      turnRight(1.0)
   elif L - R < thresh:
      # right is seeing less light than left, so turn left
      turnLeft(1.0)
   else:
      # the difference is less than the threshold, stay put
      stop()
```

Figure 27: An example light seeking Myro program.

The scribbler runs a small server program written in the PBASIC programming language. It implements a simple request-reply protocol allowing a client to query its sensors or command its motors via RS232. The scribbler server keeps some non-volatile state such as the robot's name and motor calibration data, but otherwise the server program is stateless. A software library called Myro (My Robot) written in Python interfaces with the robot. An example Myro program that performs a light seeking behavior for 60 seconds is shown in Figure 27.

The first prototype (version 0) used a wired tether. The PC would send a query via a serial port to the server running on the Scribbler and wait for a response. This wasn't

initially used in the classroom, but rather as a testing platform. However, some educators have chosen to adopt this configuration for their classroom. Once this was tested and most of the robot's functionality was exposed via its server interface we added a wireless tether. Many options were surveyed including Zigbee, a proprietary 900MHz radio, but Bluetooth was eventually chosen as the wireless tether medium.

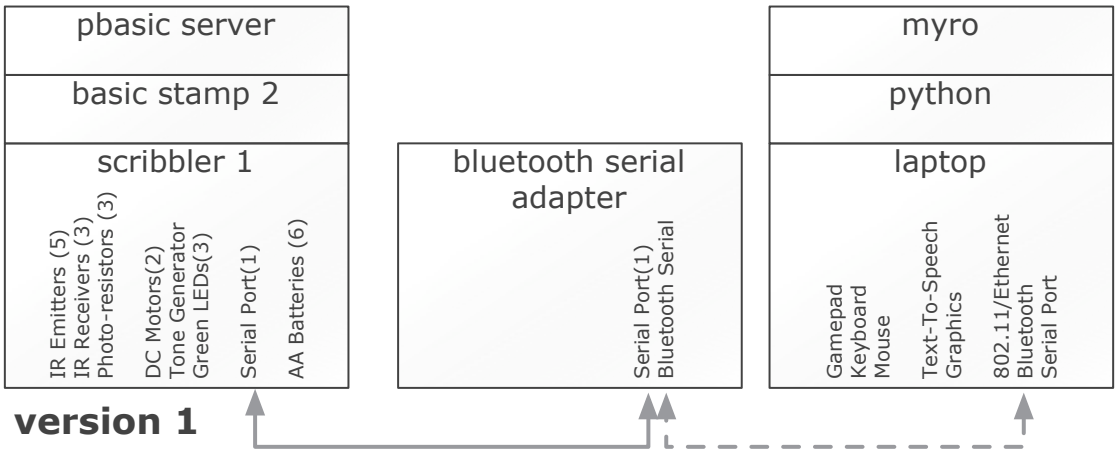
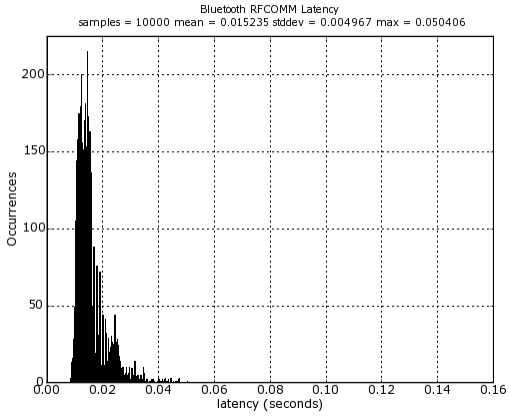


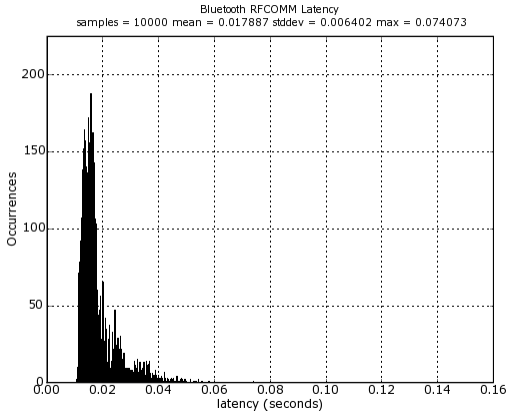
Figure 28: Version 1 of the IPRE robot system architecture

Bluetooth is ubiquitous, coming standard in many notebook computers. This also means the parts needed for adding Bluetooth to our robot were inexpensive. Additionally, all major operating systems have stable Bluetooth drivers rather than relying on some brittle driver for a home-rolled communication solution. Moreover, the Serial Port Profile and RFCOMM provide a reliable byte-stream interface which emulates a serial port. This makes the interface even more universal in terms of programming languages; most modern programming languages can interface a serial port. Bluetooth's frequency-hopping spread spectrum behavior supports many devices communicating in a shared space.

We conducted a set of experiments in order to understand the latency characteristics of Bluetooth and evaluate its viability as the tethering medium. We wrote a simple client-server program in Python that measured the latency of a request-reply interaction using the reliable Bluetooth RFCOMM protocol. Two Windows XP computers were used using Cellink USB Bluetooth Class 1 2.0+EDR USB adapters. The first set of experiments looked to evaluate the latency with different payloads at different separation distances.

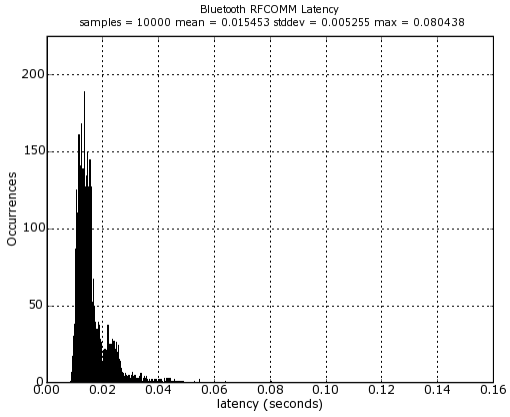


(a) 1 byte; 5 ft.

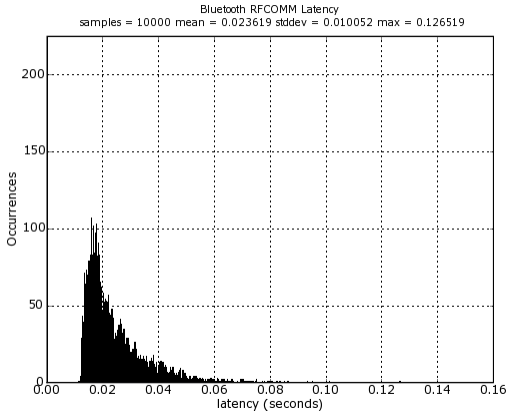


(b) 500 bytes; 5 ft.

Figure 29: Histograms of RFCOMM round-trip Latency at 5 ft. separation.



(a) 1 byte; 30 ft.



(b) 500 bytes; 30 ft.

Figure 30: Histograms of RFCOMM round-trip latency at 30 ft. separation.

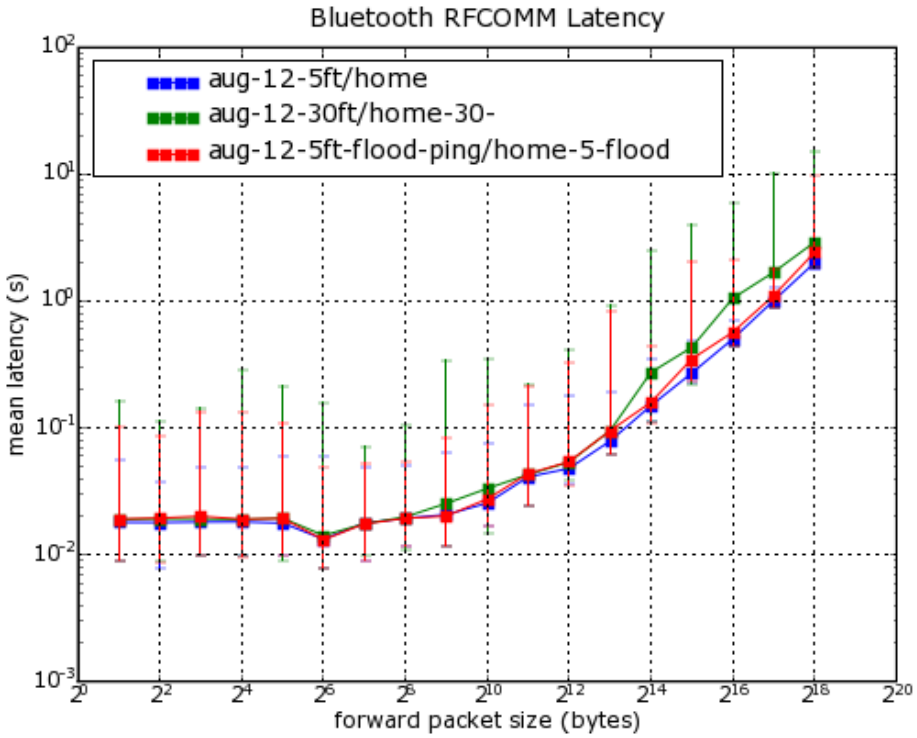
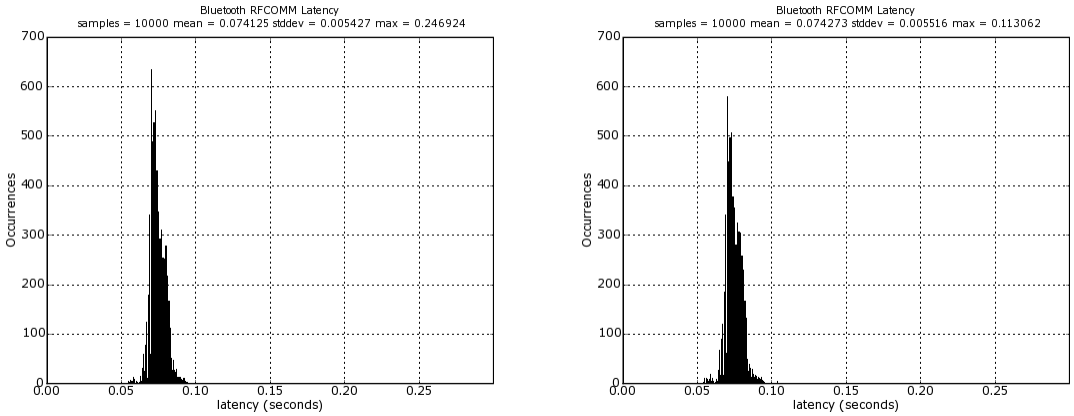


Figure 31: Mean round-trip RFCOMM latency measurements (error bars indicate standard deviation).

Figure 29 provides histograms of the latency for sending 1 byte and 500 byte payloads at 5 ft. and Figure 30 at 30 ft. separations. Ten thousand measurements were taken for each configuration. A second set of experiments varied the payload from 1 byte to 256KB. In addition, a third configuration was tested where background 802.11 traffic was generated with a flood ping. These results are shown in Figure 31.

The results confirm that the Bluetooth latency was low enough to provide responsive wireless tethering in a wide variety of configurations. On the downside, the Bluetooth protocol is rather complex and indeed was more than what was necessary for this particular system. Specifically, the initial connection procedure can be cumbersome and the latency although tolerable, is still non-zero. But in sum, the benefits outweighed the cost.

The second prototype (version 1) used the off-the-shelf Parani SD100 class-1 serial bluetooth adapter. Bluetooth devices come in three classes depending on their nominal communication range. A class-1 adapter has a nominal range of 100 meters in an open area, class-2



(a) 1 byte; 5 ft.

(b) 1 byte; 30 ft.

Figure 32: Histograms round-trip latency with scribbler at 5 ft. and 30 ft. separation.

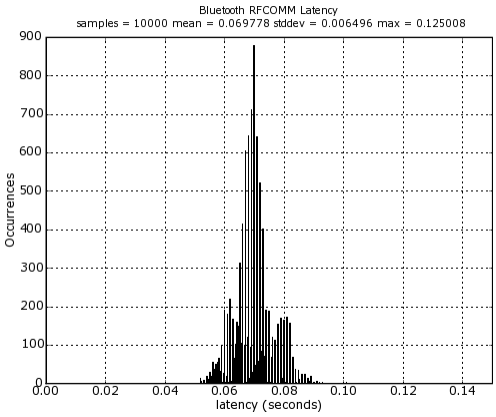
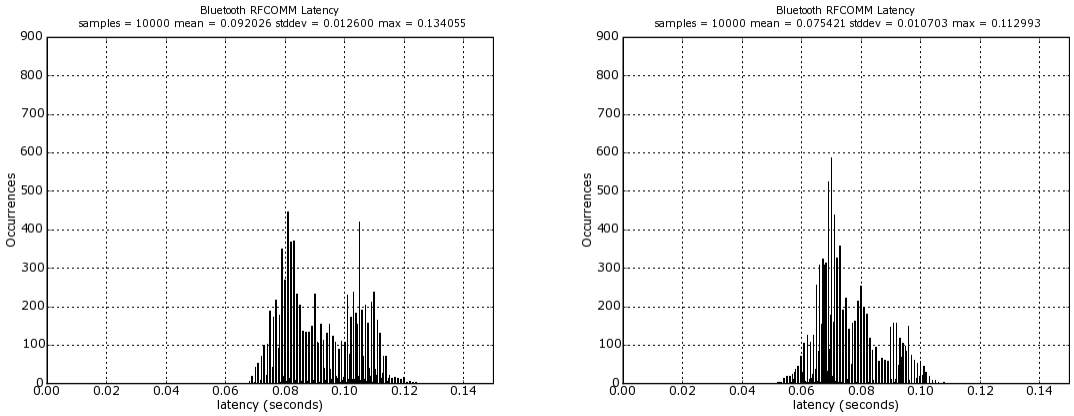


Figure 33: Histogram of round-trip latency measurements with scribbler; 6-byte return.

adapters have a range of 10 meters, and class-3 have a range of 1 meter. The SD100 adapter was powered by the Scribbler. The eighth pin on the Scribbler's serial port provided 9 volts of power. A special adapter was fashioned to connect this pin to the ninth pin of the Parani adaptor. The SD100 was effective, but it was also expensive, costing $99 in quantities of one hundred.

A similar set of experiments were run using the SD100 adapter. In Figure 32 histograms of the latency of 1 byte forward and 1 byte reply messages for 5 ft. and 30 ft. are presented. Figure 33 depicts the latency measurements of a 1-byte forward packet and a 6-byte reply which more accurately reflects the client-server operation payload. Figure 34 displays the latency when querying the robots sensors, reading the light sensors substantially impacts



(a) with light sensors

(b) without light sensors

Figure 34: Histogram of round-trip latency measurements with scribbler; 6-byte return of sensor readings.

the latency.

After three classes used the version-1 prototype (one at Bryn Mawr College and two at Georgia Tech), it was apparent the robot needed more sensors. We decided to design and implement our own bluetooth adapter, including a suite of sensors including a VGA camera and a microcontroller to interface the sensors and the Scribbler.

5.2.1 Fluke

The Fluke is a wireless robot control and sensor board (see Figure 37). Initially intended to wirelessly control the Parallax Scribbler for use in an introductory computer science curriculum, it has been used in a wider array of applications. For computation, the Fluke relies on a LPC2106 Phillips microcontroller. The ARM7-based microcontroller runs at 60Mhz with 128KB of flash memory and 64KB of RAM. An external 128KB serial flash memory is available for use by applications. The server that runs on the Fluke is programmed in a mixture of C and assembly. The GNU ARM tool-chain is used to build the software.

For communication, the Fluke uses a Class-1 Bluetooth radio which supports the Serial Port Profile. In addition, the Fluke has a RS-232 port that is nominally tethered to the robot base. Along with providing serial communication, and some mechanical support, the RS232 port can power (between 5-9 volts) the Fluke. Alternatively, the Fluke can be powered though an external power connector. The source voltage can be monitored from

```
from myro import *

def frameDifference(baseImage, newImage):
    p = copyPicture(newImage)
    for px in getPixels(p):
        bpx = getPixel(baseImage, getX(px), getY(px))
        r = abs(getRed(px) - getRed(bpx))
        g = abs(getGreen(px) - getGreen(bpx))
        b = abs(getBlue(px) - getBlue(bpx))
        setColor(px, makeColor(r, g, b))
    return p

init()
base = takePicture('gray')

film = []
for i in range(60):
    p = takePicture('gray')
    diff = frameDifference(base, p)
    show(diff, "difference")
        film.append(mi)

# save list of images as an animated gif
savePicture(film, 'diffImages.gif')
```

Figure 35: An example Myro program that performs background subtraction.

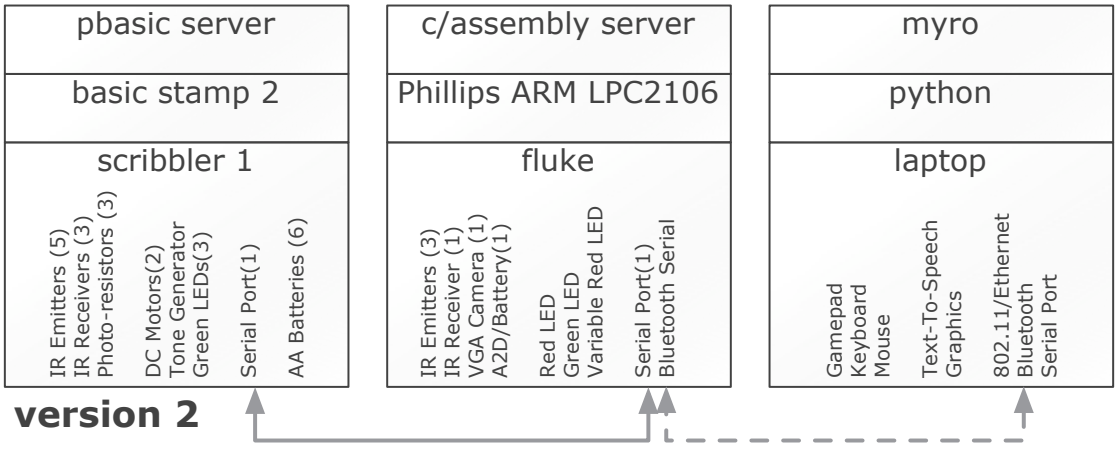


Figure 36: Version 2 of the IPRE robot system architecture

software. This was an important addition, as the Scribbler server did not have access to the battery level.

In addition to providing the wireless tether, the Fluke has additional sensors. Most

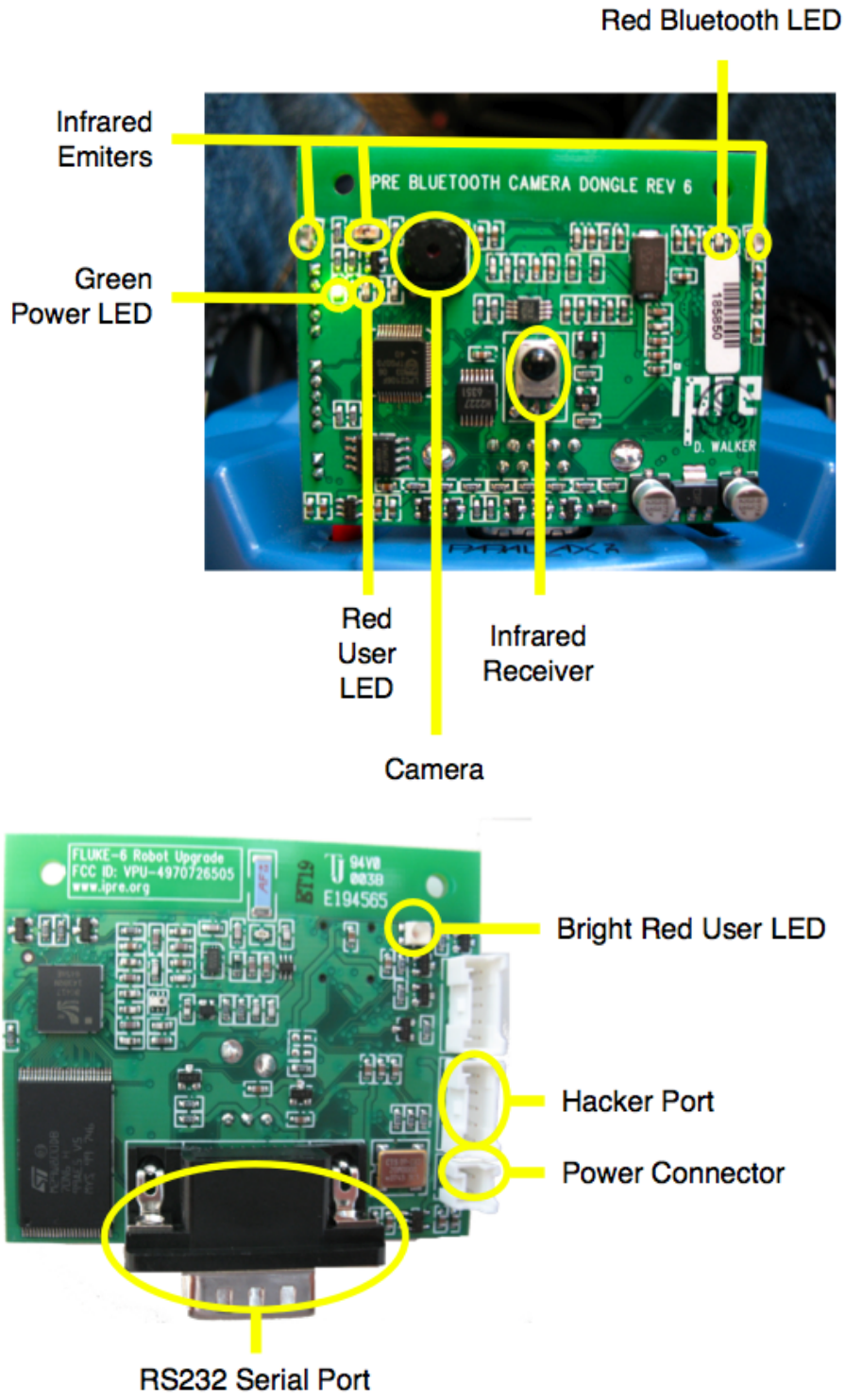


Figure 37: The IPRE Fluke

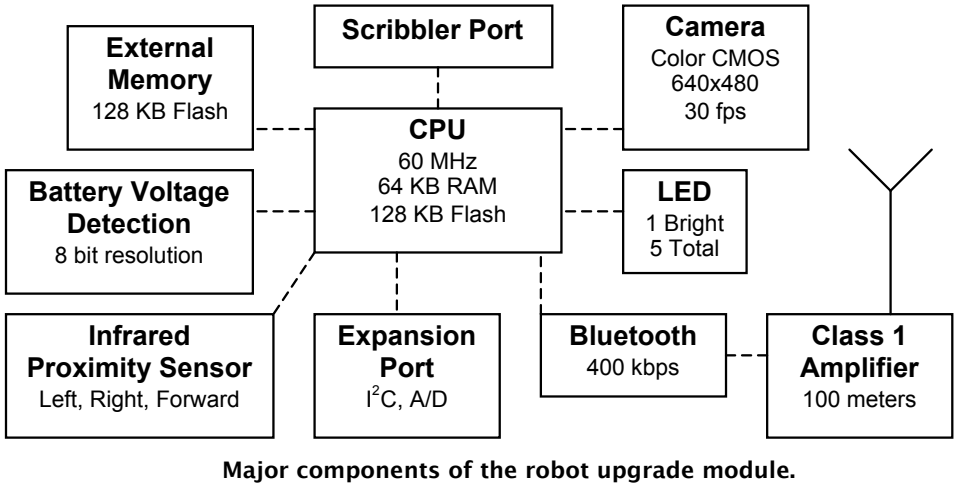


Figure 38: Fluke Block Diagram

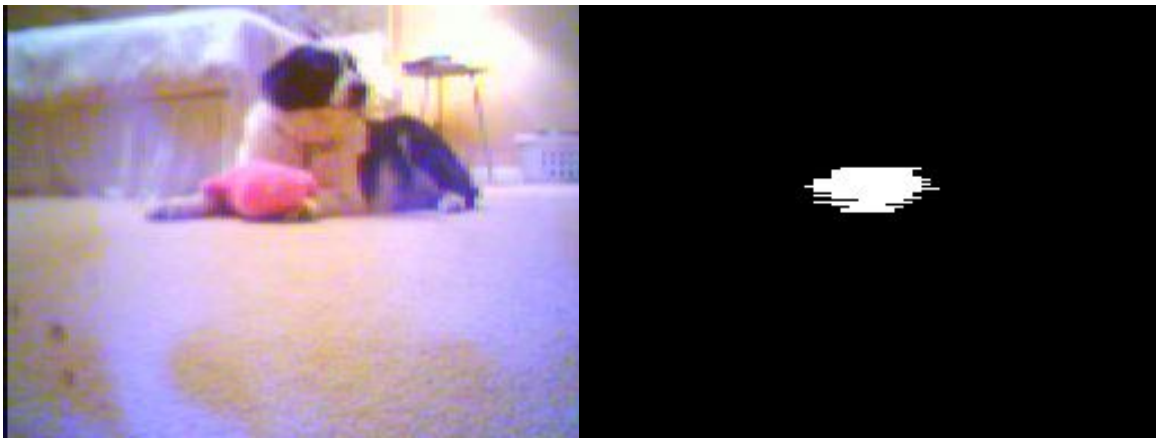


Figure 39: (a) A JPG image taken from the Fluke's camera (b) The Fluke detects pink pixels

notably, it adds a color camera (256x192 pixels). The images can be transferred over the Bluetooth link in raw or JPEG format, or processed on board using the microcontroller. Example images are shown in Figure 39. An example Myro program that performs background subtraction using the Fluke's camera is shown in Figure 35.

The Fluke has a 40 KHz IR receiver and three variable-powered IR emitters for obstacle detection and communication. The hacker port is composed of three I2C lines along with power and ground. The Fluke can reprogram its firmware dynamically over the Bluetooth link, and is capable of reprogramming BasicStamp and Propeller based robots connected to its serial port. Finally, the Fluke has one simple red LED on the camera side of the board, and a variable-powered red LED on the opposite side.

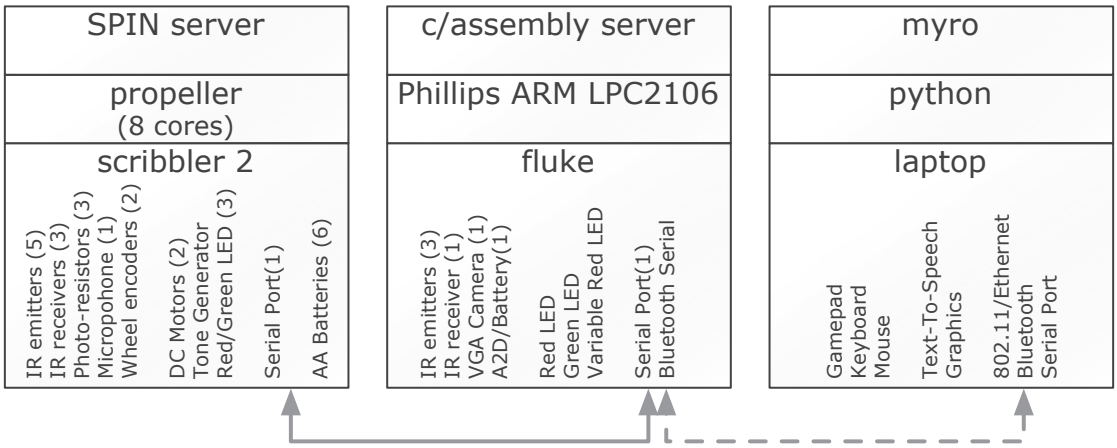


Figure 40: Version 3 of the IPRE robot system architecture

The fourth version of the IPRE system uses the Scribbler2 manufactured by Parallax. The Scribbler2 uses the Propeller processor and adds wheel encoders and a microphone to the platform.

5.2.1.1 Other Robots

The Fluke can be used without the Scribbler robot base and with other robots as shown in Figure 42. The Fluke can be run directly from a 9V battery or can interface any robot that can communication via RS232. The Fluke has been used with the BoeBot, iRobot Create, the Bioloid robot kit, the Robonova, and custom robots.

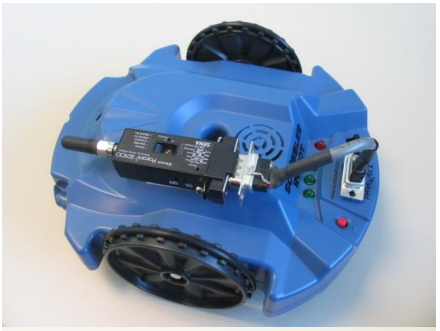
Version 0	Tethered via serial cable to a scribbler-1	
Version 1	Wirelessly tethered via an off-the-shelf bluetooth serial adapter to a scribbler-1	
Version 2	Wirelessly tethered via a custom camera bluetooth board to a scribbler-1	
Version 3	Wirelessly tethered via a custom camera bluetooth board to a scribbler-2	

Figure 41: Four iterations of the IPRE robot system

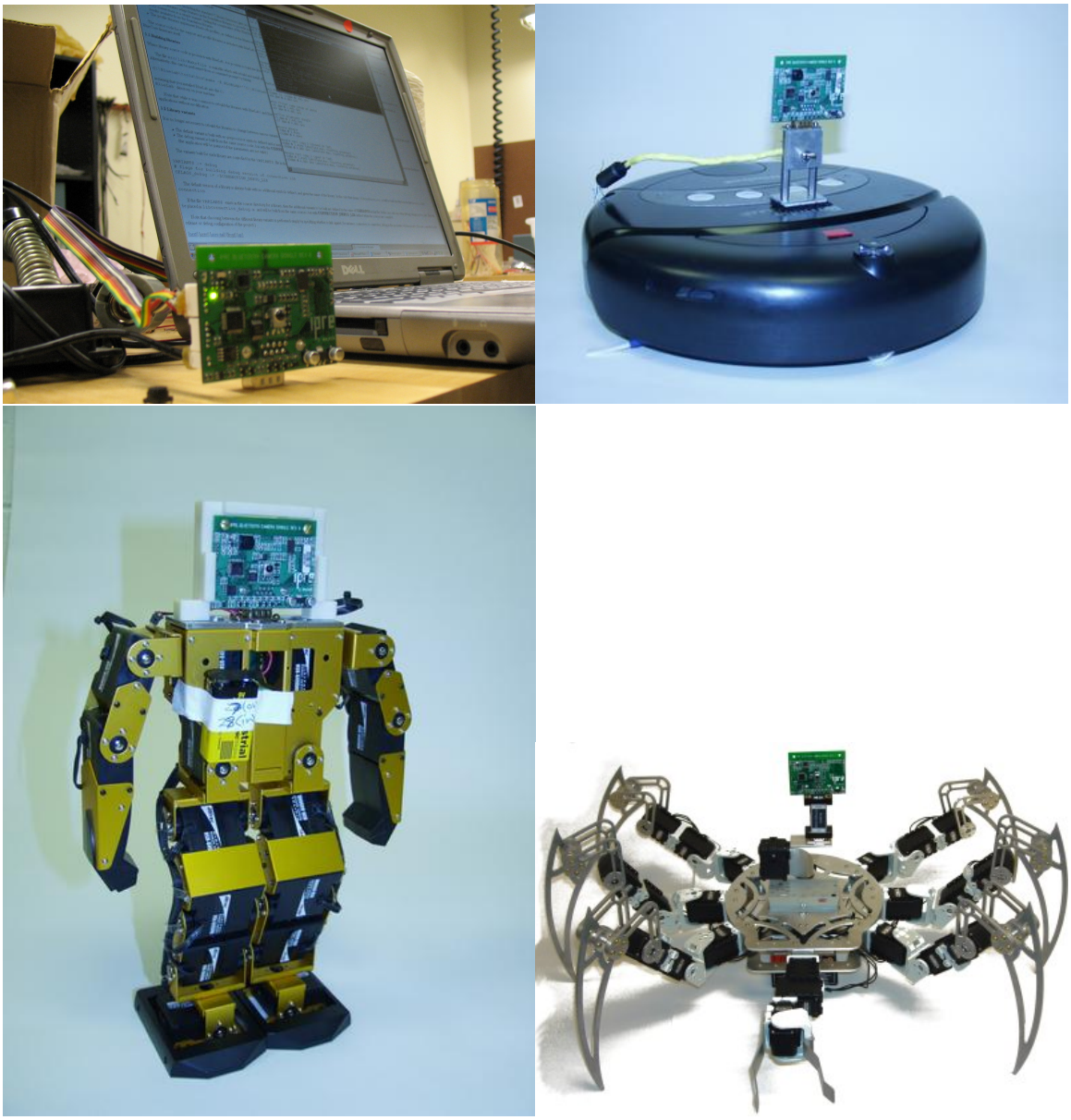


Figure 42: The Fluke has been used with a variety of robots (photo credit: Doug Blank)

5.3 Assessment

Generally, the biggest complaint from students using the IPRE system is that the robot doesn't drive straight, uses too many batteries, and is sometimes difficult to connect to using Bluetooth. The fact that the battery level can be monitored can help mitigate the last two points, and a simple motor calibration routine is used to mitigate the first. In the second revision of the Scribbler wheel encoders were to alleviate this problem. However, the Scribbler2 still relies on 6 AA batteries.

5.3.1 Introductory Computing

In order to understand how effective the robots were at engaging the students, pre- and post-surveys were conducted. To better understand how successful the students were, the students final grades were analyzed.

Table 6: Success rates of Fall 2007 classes

Fall 2007 Class	Success Rate	Students
Robots	90.97%	131 of 144
NonRobots	85.71%	78 of 91
MediaComp	73.06%	179 of 245
MATLAB	69.16%	740 of 1070

In Fall 2007 three sections of CS1301 were taught at Georgia Tech, two with robots and one without. In the two robot sections, 90.97% (131 of 144 students) were successful. The non-robots class had a lower success rate of was 85.71% (78 of 91 students). One confounding factor is that the robot class was made up of mostly computing students where the non-robots class was primarily non majors. When compared with the other introductory classes: a CS-1 that used Matlab for engineers (69.16%; 740/1070) and a CS-1 that uses Media for non-engineers (73.06%; 179/245), the robots class was very successful.

In the Fall 2007 classes, the sections shared five common final exam questions. The robots sections did 10% better on four of the five questions than the non-robots section. The difference was statistically significant ($p < 0.015$).

5.3.2 Teaching Shortest Paths

Although our educational robot is indeed a distributed robot system already, we would like to further understand how multiple robots can be used to teach more advanced concepts. Moreover, we believe a significant contribution of our Gnats system is its simplicity. Although, proving simplicity and minimalism is often an arduous tasks, we believe that showing the system can be understood and built-upon by novices is strong empirical evidence of our simplicity claim. Thus, we used our Gnats system to teach the concepts of robot navigation, and more abstractly, shortest paths through graphs. Our hypothesis was that the robotics context will not only give a real-world example of this abstract problem, but the use of multiple robots by the Gnats system, with each robot taking on the role of a node in the search space, gives an intuitive, physical illustration of the abstract notion of space space, resulting in better general understanding.

Two groups of students were taught about shortest paths as it relates to navigation. The first group were the control group and were taught about shortest paths in a traditional way. An algorithm based on breadth-first search from the start location was presented as a way to find a minimal cost path to some goal location. The algorithm was presented in the context of navigating from one state to another on a map of the U.S. Python code was used to explain the algorithm, but using a typical graph-based explanation.

The second group, the experimental group, was taught about shortest paths using the Gnats algorithms. In addition to describing the Gnats algorithms and previous experiment, a live demonstration was used.

Table 7: Results of the shortest path post-survey and assessment problems.

	Control	Experimental
Number of Students	18	47
"I Understand Shortest Paths"	2.94	3.02
"I Enjoyed this Lecture"	3.02	2.66
Robot Navigation Problem 1	89%	96%
Robot Navigation Problem 2	72%	74%
Social Network Problem	56%	48%

After the lecture, a short written survey along with three shortest path problems were

completed. The first set of questions had the students work through robot navigation problems, and the second section required the students to apply their knowledge to shortest path problems in another domain, social networking.

The robot group claimed to understand shortest paths algorithms slightly more than the control group, but there were no statistically significantly differences between the two groups. The results are summarized in Table 7. For the survey questions, a likert scale was used (1-5) and the mean responses are reported. We also analyzed the percentage of the groups that successfully completed the assessment problems. Of the robot group, 96% of the students successfully completed the first robot navigation problem, and only 89.0% of the control group correctly solved it; however, a greater proportion of the control group solved the social networking problem. None of the results were found to be statistically significant.

5.4 Reflection on Architecture

We have leveraged distributed robot system design in order to create a personal robot for computing education. The distributed robot system is composed of an inexpensive mobile robot augmented with a Bluetooth sensor board, and a laptop for computation and additional input and output modalities. This robot-as-peripheral approach offers many advantages over the compile-down-and-run model in terms of robustness, ease-of-use and expressiveness, while minimizing cost.

5.4.1 Lessons Learned

- "Personal Robots" means "Plentiful Robots"
  - physical interference; lunch-box size works nicely
  - amplification of communication problems; bluetooth works nicely
  - amplification of usability problem; bluetooth works not-so-nicely
- No assembly required; but don't prohibit personalization
- Personal robots; but not isolated robots

- If you can't afford it, fake it

CHAPTER VI

DESIGN PRINCIPLES FOR ROBOT SYSTEMS ARCHITECTURE

Like computer architects, robot designers must address multiple, possibly competing, requirements by balancing trade-offs in terms of processing, memory, communication, and energy to satisfy design objectives. For example, architects might strive to minimize the energy use or cost of a memory subsystem, or maximize the reliability or availability of a storage system. However, unlike computer architects, robot designers have the additional dimensions of sensing and actuation to consider. Robots live in the real world, sensing and effecting external physical phenomena. This leads to a key consideration that is sometimes lost in traditional computing system design. Where is the robot computing system located in physical space? This consideration amplifies the role distribution plays in robot system design. The allocation and organization of the sensing, computing, actuation, energy, communication resources throughout physical space is at the core of distributed robot systems architecture. The physical distribution of resources let's us exploit the locality of the particular task in a similar manner as the design of a memory subsystem let's us exploit the locality of a computational problem. Robot architects currently lack the design guidelines, organizing principles, rules of thumb, and tools that computer architects rely upon. This thesis takes a step in this direction, by analyzing the roles of heterogeneity and distribution in robot systems architecture.

6.1 Task and Architecture

Traditionally, robots have been designed with very a clear task in mind, for example, robot arms assembling automobiles. This is in contrast to personal computers that are designed for a wide variety of applications. Because personal computers can be reprogrammed to perform almost any task, they are designed in a way to accommodate many different applications. Similarly, as robot systems become more general purpose, designers will not have the luxury of a single specific task objective. Instead, robots will be reprogrammed and reconfigured

Table 8: Summary of Architectural Concepts in Computer and Robot Design

Architectural Principle	**Computer Architecture Example**	**Robot Systems Architecture Example**
scaling laws	Amdahl's rule of thumb balanced architecture	robot architecture evolution
storage mechanism	random vs. sequential access	pervasive vs. mobile access
locality	spatial locality temporal locality	physical spatial locality physical temporal locality
boundedness	CPU-bound; memory-bound; IO-bound	sensor-bound; actuator-bound; communication-bound

for their particular task.

For some tasks, like robots exploring Mars, it's plausible to design a robot system from start to finish for a specific task or handful of tasks. However, for more commercial systems, it will be more economical and flexible to mass produce more general purpose robots that later can be reprogrammed and reconfigured. Therefore, although we might be able to design a robot system from beginning to end, and in fact, maybe design it optimally in some sense, there are few situations where this is economical. Therefore, we need to identify classes of applications we expect the robot system to address and design accordingly.

6.2 The Access Mechanism

As an example of a robot systems architecture notion, consider two classes of robot computing systems: mobile robots and sensor networks. Both types of systems can be used to solve many of the same robot computing problems (e.g. environmental monitoring), but there also exist applications for which each is particularly suited. Both mobile robots and sensor networks provide access to large-scale phenomena – mobile robots via mobility, and sensor networks via pervasiveness. Thus, we term both mobility and pervasiveness as "access mechanisms".

A large class of robot computing tasks that are suitable for both mobile robots and sensor networks fall under the heading of "monitoring". For instance, consider tasks such as forest fire detection, military reconnaissance, security patrolling, and urban mapping. In these tasks, we want to collect sensor data from a large area and distill it down to the

information of interest. Any of the 4-D's of robot tasks: *dirty, dangerous, difficult, and dull* may make robotic monitoring useful.

Whether to use, and how to use, mobility or pervasiveness as the access mechanism is one of the most interesting robot architectural trade-offs. We term it the "access trade-off". One mechanism may be more suited for the task, or some mixture of both. After all, mobile robots can benefit from pervasiveness, likewise, sensor networks from mobility. Rather than just building a fully mobile sensor network or a fully pervasive robot swarm to exploit these benefits, this thesis takes a more nuanced approach. Our architectural approach enables us to exploit certain aspects of the system, as seen as an more abstract robot computing system in terms of energy, reliability, and performance.

6.3 Architectural Principles

Three concepts, **boundedness**, **locality**, and **balance** from computer systems design are useful for describing robot systems from an architectural point of view. These ways of thinking about computational processes abstractly can be leveraged in the design of robot computer systems. Although these notions are abstract they aren't theoretical, a real, concrete computational process must be involved. In a similar manner, these are properties of a robot computational process and involve a robot systems architecture performing some task.

- **Locality** – a useful way for describing the memory access behavior of a computational process. Similarly, we can adapt this notion to robotic systems to describe the environmental access behavior of a robot process.

- **Boundedness** – a useful way for describing the resource utilization behavior of computational processes. Different computational processes might spend a majority of their time processing, doing input or output, or accessing memory; a computational process can be CPU-bound, IO-bound, or memory-bound. The same idea can be applied to robot systems architecture. A robot process might be bound by its sensor bandwidth, the amount of CPU processing it has, or by the speed of its wheels.

- **Architectural Balance; Scaling Laws** – useful concepts for computer systems design and evolution; a way of describing and comparing different robot systems' resource allocation.

6.3.1 The Locality Principle

In computer architecture[45], the concept of "locality of reference" is crucial in the design and organization of a computer system's memory, informing things as diverse as the design of virtual memory systems and web caches, and the hardware organization of cache memories[28]. The locality of a computational process characterizes its memory access behavior. If a computational process has a high level of spatial locality then memory accesses are local in terms of space, i.e. nearby memory locations are more likely to be accessed in the future. Likewise, if a computational process is temporally local then the same memory locations will be accessed often. Things like web caches exploit the temporal locality of web browsing behavior, i.e. storing web pages that are likely to be accessed again. Pre-fetching or read-ahead buffering are mechanisms to exploit spatial locality. Similar concepts can be used to describe robot processes and in the design of robot computing systems.

We define the concept of "physical locality" to be the environmental access behavior of a robot process. If a robot process has a high level of spatial locality then nearby locations are more likely to be accessed in the future. Likewise, if a robot process is temporally local then the same locations will be accessed often. For instance, if a robot task is temporally local, but not spatially local – we need to monitor distant locations frequently – then a pervasive sensor network is well suited for the problem. Reciprocally, when the robot process is spatially local, but not temporally local – we visit environmental regions in a "sequential" manner, but rarely visit the same point twice – a mobile robot is well suited for the problem. Therefore, the locality of a robot process can help us decide what kind of architecture is more suitable.

Both distribution and heterogeneity can exploit the locality of a particular task. Distribution let's us place our resources close to where the robot computation is happening, and heterogeneity let's us place specialized types of resources at different locations.

6.3.2 Boundedness

In addition to describing a computational process by its access pattern, its locality, we can also note what percentage of time, energy, or money the task devotes to computing, accessing memory, sensing, or effecting the environment. The boundedness of a task is the limiting factor. For instance, the performance (measured in time til completion) of a memory-bound task is limited by primarily by memory access times.

The notion of *boundedness* from computer systems design is a useful concept for thinking about robot tasks. Different computational processes might spend a majority of their time processing, doing input or output, or accessing memory; a computational process can be CPU-bound, IO-bound, or memory-bound. If a process is memory-bound then we can identify the memory-subsystem as the bottleneck. If we plan to improve the system, we are wise invest on improving that bottleneck rather than the other areas.

We can apply the same idea to robot systems architecture. A robot process might be bound by its sensor bandwidth, the amount of CPU processing it has, or by the speed of its wheels. A robot that is performing a batch mapping task might be limited by its laser scanning bandwidth. This notion also helps us identify, and correct, bottlenecks in robot system architecture. This idea enables us to group different tasks according to the how they are bounded.

6.3.3 Balanced Architectures

Although Gene Amdahl is typically known for his law[4] of diminishing returns concerning parallel computation, his rule of thumb[42] (also known as his other law) is concerned with the architecture of computer systems. It says that for a computer to be useful, for every instruction per second of computation, the computer must also have one byte of memory, and one bit per second of IO. His rule of thumb provides a scaling relationship between the constituent parts of a computer system, and also the notion of a balanced computer system. Similar questions can be asked about robot computing systems. For instance, for a household mobile robot, what is a useful proportion of computing to memory, to sensing, to actuation? Although, a full answer to this question is beyond the scope of this thesis, we

do point out that a "balanced" robot architecture does not have to mean equally balanced, but rather just some desired proportioning of resources.

Amdahl's rule of thumb gives us a new way to compare and reason about different robot systems: how are the systems' resources allocated? Is the robot equally-balanced, with its resources equally allocated across sensing, computing, and effecting or is the robot rather asymmetric, for example, with sensing dominating? Some tasks, might not require symmetry, in fact they might benefit from specialized, asymmetric architectures. However, one could argue that for general purpose robots intended to provide a wide variety of applications, much like general purpose computers, a balanced architecture is beneficial. Generally, striking a balance in terms of architecture can help alleviate architectural bottlenecks.

We can arrive at a balanced robot system in a variety of ways. One possibility is for an individual robot to be balanced from its initial1 design. Another possibility is to achieve balance through composition. By connecting a collection of specialized platforms, a more general system results. The idea of using distribution to achieve a *balanced* system architecture is a theme of this thesis. For example, in designing a robot for education we arrived at a balanced, general-purpose platform from a collection of specialized platforms. This particular platform which is composed of a mobile base (the scribbler), a bluetooth sensor board (the fluke), and a laptop. The laptop is very computation centric, the scribbler mostly focused on mobility (and thus actuation), and the fluke on sensing. The ternary diagram in Figure 43. visualizes how the individual specialized platforms allocated their resources in terms of cost to the sensing, computing and effecting subsystems.

6.3.4 Robot Design Constellations

The use of ternary diagrams to reflect on an architecture's balance and to compare different robot designs is useful in understanding robot systems; however, it only allows us to compare along three dimensions. In Figure 43, we chose to compare the robot platforms in terms of the budget allocated toward computing, sensing, and effecting. To move beyond only three dimensions, we have adopted the star (or radar) chart [47] for describing robot system architectures. As noted in [47] star charts are particularly good for comparisons, the authors

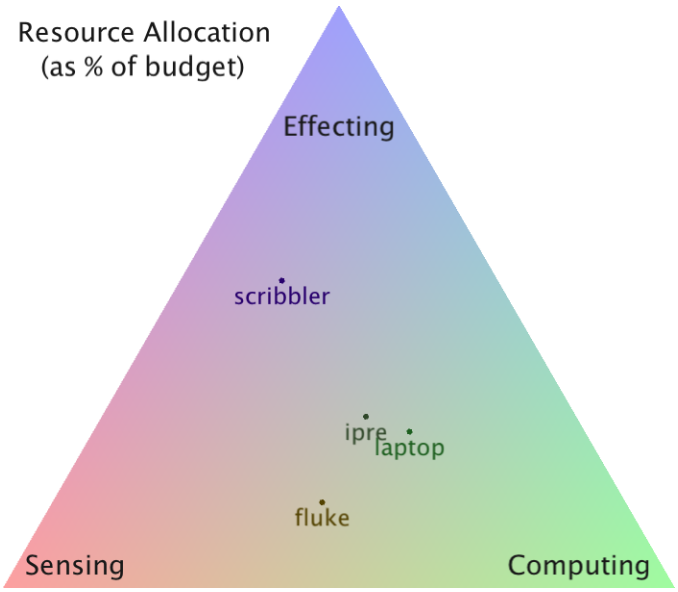


Figure 43: The resource allocation of the individual specialized platforms of the IPRE robot system.

remark, "one of the main purposes of such multi-code schemes (like star charts) is to obtain a symbol with a distinctive shape for each observation, so that a viewer can look for pairs or groups of symbols with similar shapes, or individual observations that are very different from the rest."

We use the term "robot design constellation" to describe the use of star charts to visualize a collection of connected robot platforms. In Figure 44 we visualize the IPRE system for computing education as a design constellation. Again, this diagram clearly illustrates the platform's "balance", and how each of the constituent platforms differ. A similar platform the Finch[55] is visualized in Figure 45 for comparison. Notice that the Finch which foregoes a camera, puts less of its budget in its sensing subsystem. Figure 46 visualizes the Gnats systems and as a comparison, Figure 47 visualizes the Kilobot swarm robot[90] in comparison to the Gnats. The Kilobot allocates more resources to mobility. Figure 48 visualizes the Autopower system.

Design constellations not only support comparison between platforms, but also allow architects to interactively explore design trade-offs. In describing star charts, Chambers et al. [47] suggest that "It can be helpful to cut the symbols apart (along with their labels) onto separate slips of paper, and to slide them around on the table grouping them in interesting

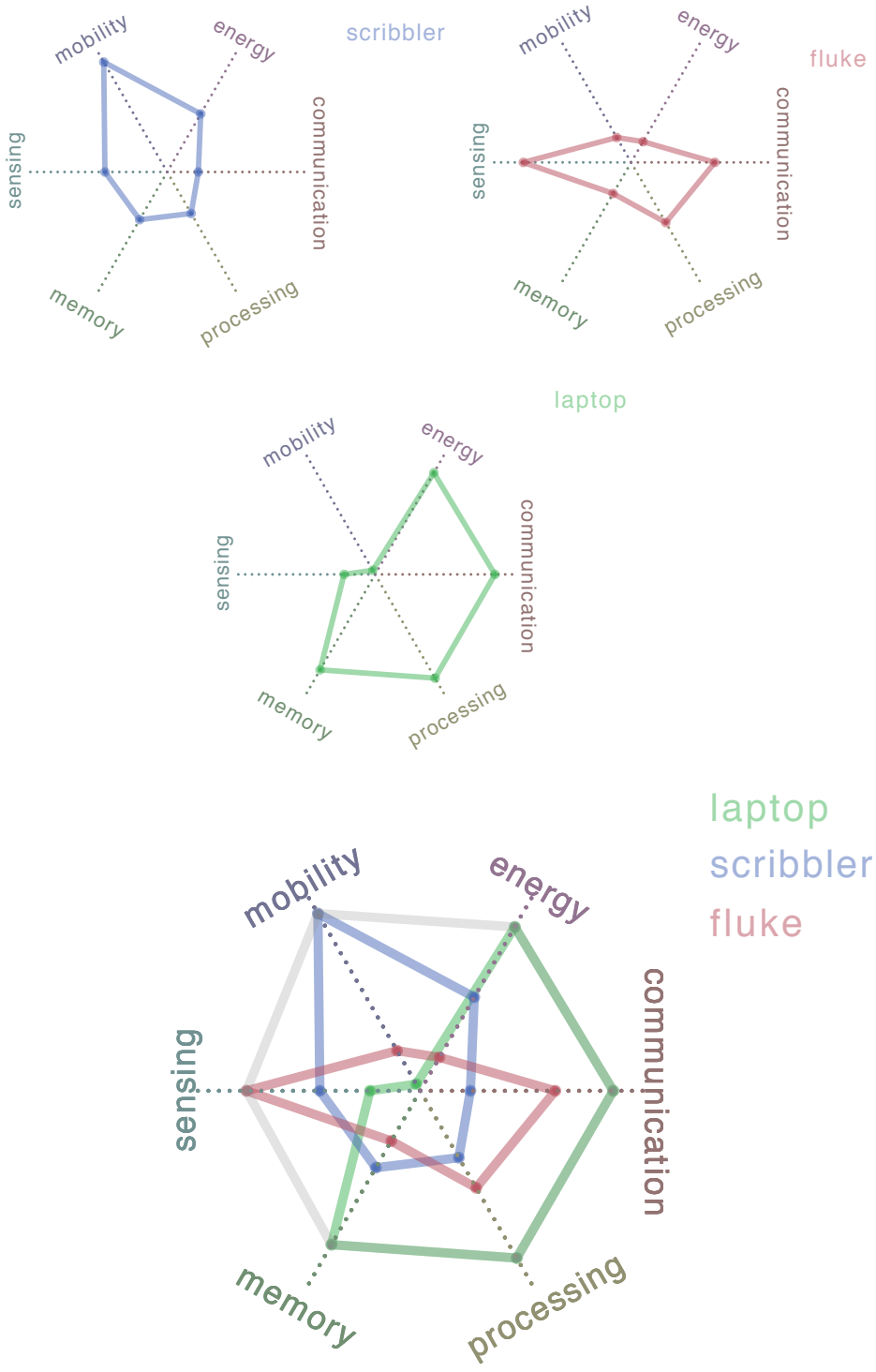


Figure 44: Design constellation of the IPRE robot system for education.

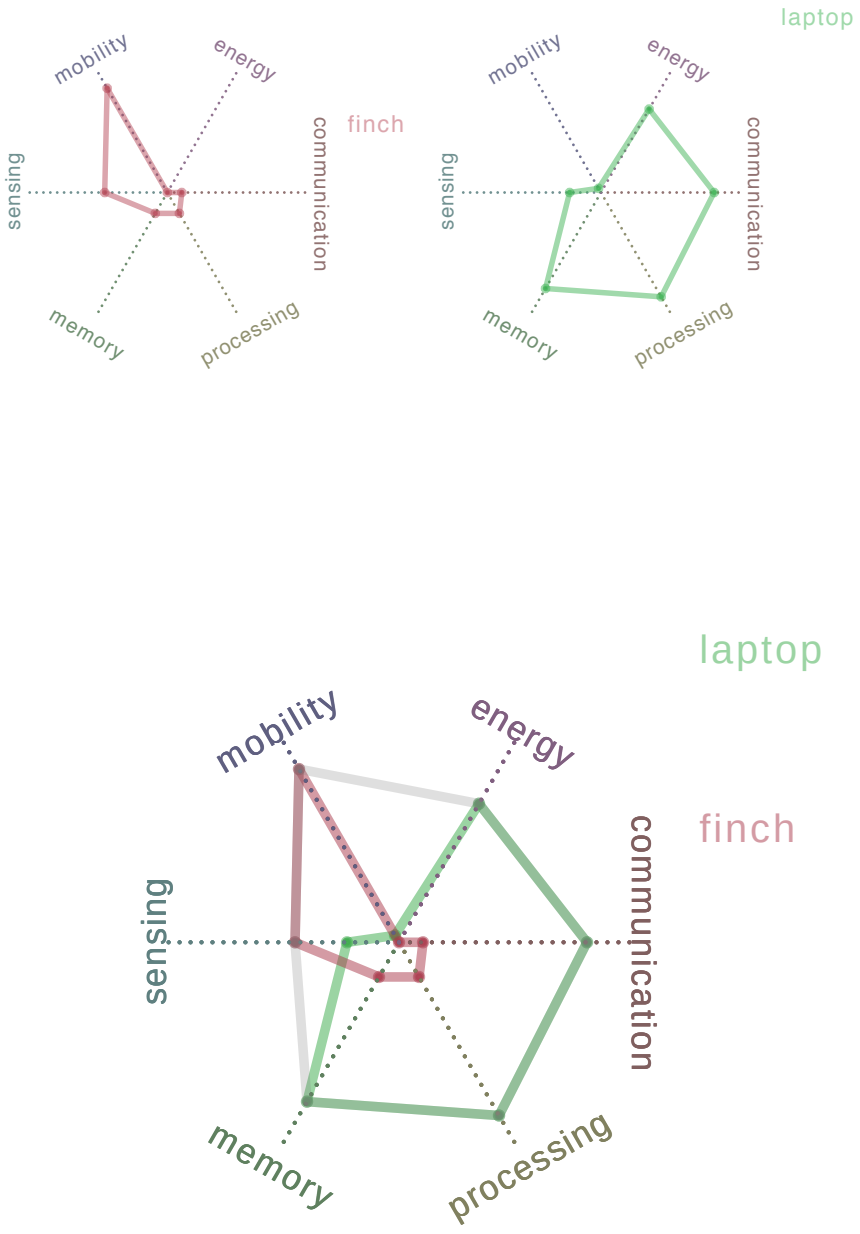


Figure 45: Design constellation of the Finch robot system for education.

ways." We created a program to construct these star charts and *slide them around* as the authors suggest to support comparisons. A program (a Processing sketch) was implemented

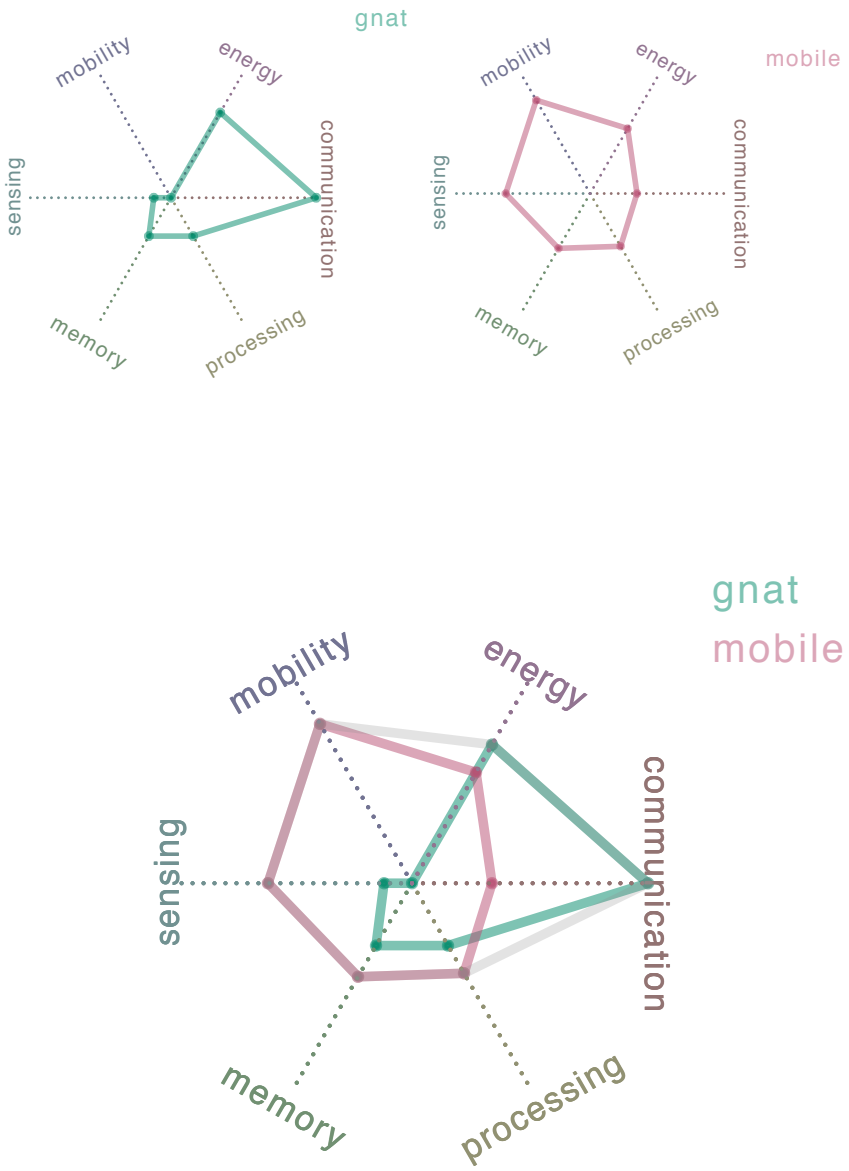


Figure 46: Design constellation of the Gnats robot system.

to allow architects to interact with the constellation in order to explore trade-offs. The user can add (or subtract) resources to (or from) a particular dimension and see how the overall platform is impacted.

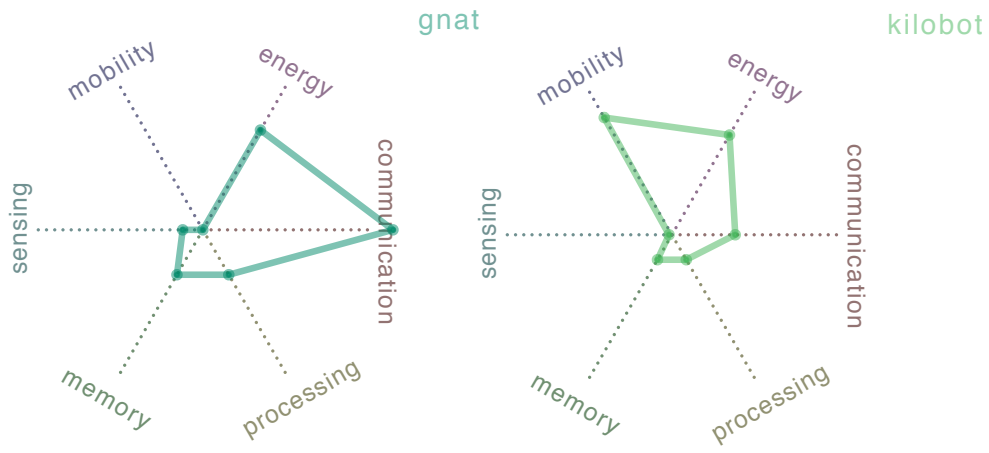


Figure 47: Design constellation of the Kilobot and Gnats robot systems.

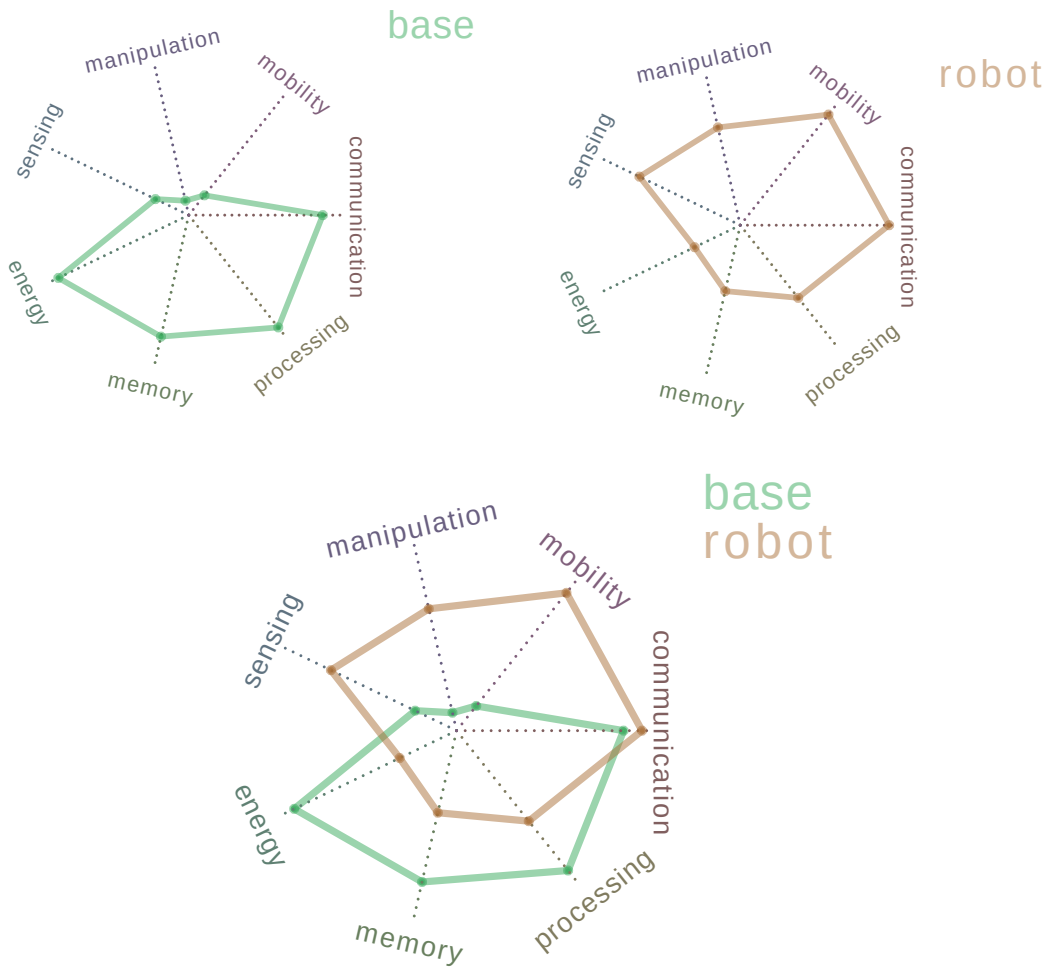


Figure 48: Design constellation of the AutoPower system.

CHAPTER VII

CONCLUSION

This thesis takes a systems architecture approach to the design of robot systems, and in particular, investigates the use of distributed, heterogeneous platforms to exploit locality in robot systems design. We show how multiple, distributed heterogeneous platforms can serve as general purpose robot systems for three distinct domains with different design objectives: increasing availability in a search and rescue mission, increasing flexibility and ease-of-use for a personal educational robot, and decreasing the computation and sensing resources necessary for navigation and foraging tasks.

Through the exposition of three robot systems that exploit heterogeneity and distribution in novel ways, this thesis offers not only lessons learned, but some general design guidelines for using distributed robot systems to achieve various design objectives.

All robot systems are inevitably going to be distributed systems, and the earlier this is taken into consideration the better. Moreover robot systems can be constructed from a collection of specialized robots that are both sufficiently general to solve the task and sufficiently specialized to exploit the task's locality.

7.1 On the Disadvantages of Distribution

> "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable" – Leslie Lamport

Although this thesis argues and provides evidence for a distributed approach to robot system design, of course, this approach has disadvantages. Most importantly, when functionality is distributed across multiple machines the likeliness of an overall failure increases. Failures in individual platforms or the interconnect, or interference between platforms can be problematic, for instance:

- **Gnats** – If the embedded network becomes disconnected, for example, an embedded

node fails and disconnects the network into two connected components, the algorithms for path planning, coverage, and foraging break. The mobile robot only learns of the environmental configuration by communication and thus can only access the environment being covered by its current connected component.

- **AutoPower** – The AutoPower system is vulnerable to interference effects. Since all the mobile robots are relying on the base station for computational and energy offloading, if any of the mobile robots monopolize that resource (e.g. by way of a software defect or an inaccurate energy behavioral model) the rest of the team can be negatively impacted.

- **IPRE** – Students sometimes have trouble establishing the initial bluetooth communication connection between the laptop and the mobile robot. And without this communication link, the mobile robot is useless to the student since the laptop provides all the user interaction.

When building any distributed system, including the robotic variety, it's important to remember Peter Deutsch's[1] fallacies of distributed computing.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

[1] `http://blogs.oracle.com/jag/resource/Fallacies.html`

7.2 Detailed Contributions

Distributed Navigation and Foraging [68, 69, 70, 71, 73]

- A path planning technique using a minimal embedded network rather than using traditional mapping and planning is designed and implemented.
- A technique for supporting multi-robot foraging is designed, implemented, and analyzed in terms of sensitivity to deployment and environmental conditions.
- The path planning algorithm is implemented on a real multi-robot system and the quality of the resulting paths is investigated.
- Evidence is provided that a mobile robot can effectively use the Gnats' paths in static and dynamic environments.

Energy-Aware Search and Rescue[72]

- A system-level model of software energy behavior and platform energy characteristics is developed.
- Software allocation and connectivity decisions are shown to prolong the lifetime of a robot team.

Personal Robots for Education[8, 97]

- A personal, distributed robot system for teaching introductory computing is designed and implemented.
- Evidence is provided for how a multi-robot system can aid in the learning of computing concepts.
- Evidence is presented that students are able to apply shortest path algorithms to navigation problems effectively after learning about the Gnats algorithms.

Robot Systems Architecture[67]

- The principles of locality and balanced computer architectures are applied to the study of robot system architecture.
- "Robot Design Constellations" are introduced as a means to explore design trade-offs and support comparison of robot platforms.

7.3 Future Work

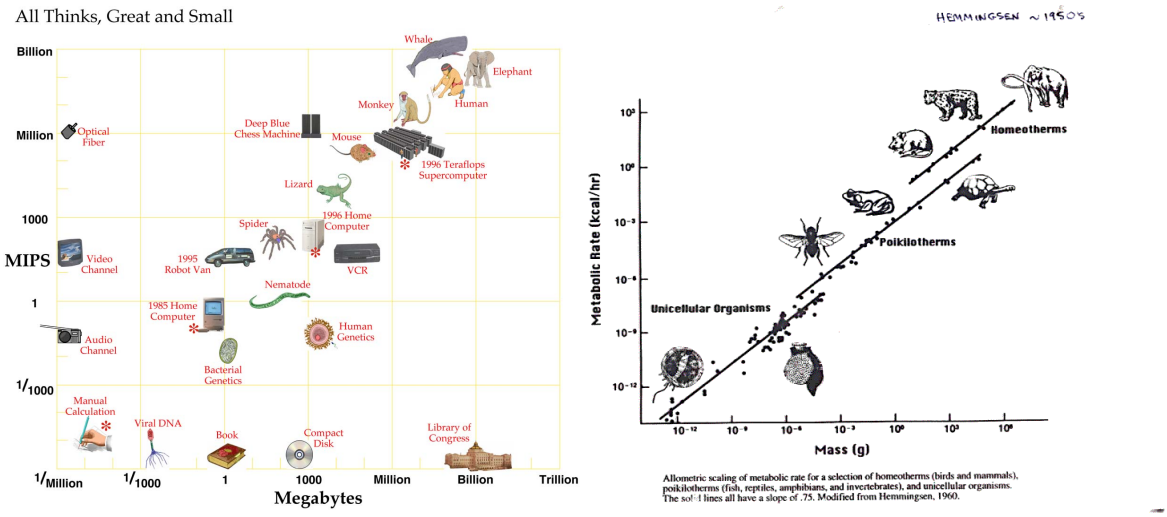
Currently, the architectural design of robots is more of an art than a science. Robot architects currently lack the design guidelines, organizing principles, rules of thumb, and tools that computer architects rely upon. This thesis takes a step in this direction, by analyzing the roles of heterogeneity and distribution in robot systems architecture.

Robot systems architecture has the potential to live between the current artful practice of robot design, which often starts from scratch with each new design, and the holy-grail goal of fully automated system design.

7.3.1 Scaling Laws

Gene Amdahl's rule of thumb is concerned with the architecture of computer systems. It says that for a computer to be useful, it needs to be balanced. More specifically, for every instruction per second of computation, the computer must also have one byte of memory, and one bit per second of IO. Hans Moravec[64] restates Amdahl's rule of thumb to include animals as well as machines (shown in Figure 49(a)). Amdahl's rule of thumb is really a "scaling law", a term familiar to biologists and mechanical and aerospace engineers. Amdahl's rule of thumb describes how the relative growths of the architectural components are related for a balanced computer system; however, we currently have no analog in the design of robot computing systems. It is unclear for a household mobile robot, what the proportion of sensing to computation to memory to actuation ought to be. Moreover, what should the proportion of computing, sensing, and actuation be to physical quantities such as area, volume, mass, speed, and acceleration?

Although some recent theoretical approaches[74] offer general models of what a robot



(a) Moravec illustrates Amdahl's rule of thumb as applied to animals and machines.[64]

(b) How metabolic rate changes with body mass [44].

Figure 49: Scaling laws in animals and machines. (reproduced without permission)

needs to complete certain tasks, their generality also limits their applicability to real systems. In contrast, an empirical approach to understand the underlying space of robot architectures could be based on real fielded systems. Scaling laws could be uncovered by surveying fielded systems in terms of the computing, sensing, and actuation resources they used for different classes of tasks. The scaling laws would inform robot designers on how to build "balanced" robots for classes of tasks rather than for point applications. It will also further robot science in terms of an empirical understanding of the underlying information requirements[29] of different robot tasks. Moreover, an understanding of the true scaling nature of robot systems would help determine the feasibility of pushing robot systems to massive levels (e.g. robot swarms or smart dust) and enables us to build high-fidelity, small-scale experimental systems.

Biologists use scaling laws to describe many different natural phenomena, a method referred to as "allometry". The work by D'Arcy Thompson[99] is widely accepted to be seminal in allometry. He put forth a very simple, and somewhat controversial, quantitative method for describing how animals' bodies scale. Scaling laws are used pervasively throughout Biology. Brooke et al. discuss how the size of a bird's eye changes with its body mass[26]. Stevens[95] discovers a $\frac{3}{2}$-power law relating the number of neurons in the

visual cortex of primates to the number of thalmic neurons (which is proportional to the number of retinal ganglion cells). A recent book edited by Brown and West[16] highlights some recent results in biological scaling. Figure 49(b) relates how an animal's metabolic rate changes with its mass[44].

We can borrow the natural science methods of taxonomy, survey, and allometry for the artificial, yet physical, science of robotics. To uncover scaling laws for robot systems, we plan to conduct a large-scale survey of real, fielded, robot systems. For instance, for humanoid robots (which imply a class of actuation and mobility mechanisms), what levels of sensing and computing were required? For laser scanner based ground navigation, what levels of computation were required? Does this differ from aerial vehicles? How did computing, energy, and sensing scale for mobile robots versus sensor networks? How did sensing and computing scale with the physical size of the robot?

The survey would include on the order of hundreds to thousands of robot computing systems detailing their computing, communication, sensing, actuation and mobility capabilities. Rather than conducting the survey manually (like those poor *natural* scientists), we could build an automated system to mine the published databases (e.g. Google Scholar) for relevant articles describing fielded systems.

7.3.2 Locality and Access Mechanism

It is hypothesized in this thesis that the physical locality principle can greatly inform the system architecture of robot systems. Looking forward, further exploring how this locality property can affect design decisions such as the access mechanism is a ripe area of research. Moreover, quantitatively capturing the notion of locality, as we can do in computer architecture, is an exciting direction of research.

In addition, the idea of physical locality and access mechanism seems useful outside of robotics, for example, in understanding various type of networking. For instance, it seems directly applicable to areas like delay-tolerant networking[48] where mobility is used rather than pervasiveness for communication.

7.3.3 Embedded Networks and Mobile Robots

The Gnats, and embedded networks in general, can support many types of mobile robot applications. They can aid in mapping, localization, coverage, navigation, task allocation, traffic coordination, and many other applications. By adding sensors or more computational power to the devices, we can enable even more applications. In addition, the embedded network can be supported by mobile robots. The robots can deploy and maintain these large networks.

The current path planning algorithm assumes the network connectivity information is the same as the environmental connectivity. An improvement would be for the network connectivity to act as an initial approximation of the geometric connectivity, which then could be refined by real robot navigation experiences. This would allow robots to discover obstacles not sensed by the network. Also, a variety of path following algorithms could be developed for a mobile robot to navigate through the embedded network.

Using the network for multi-robot coordination is a very interesting avenue of research. Imagine large systems of 20-100 mobile robots. Explicit coordination between all the mobile robots will be troublesome and might consume most of their resources, leaving little time to actually complete the task at hand. Instead, the embedded network could be responsible for this coordination. For instance, the network could provide task allocation services to the robots. Another example is traffic management. Consider applications where multiple robots have to coordinate their paths, for example, in a foraging or delivery task. Rather than trying to plan paths in a very high dimensional space, the mobile robots can use the embedded network as an ad-hoc traffic system.

The Gnats algorithms for coverage, navigation, and foraging assume a mostly connected network. This may not be the case if there are failures or interference. Mobility, even in a very limited sense, can be a useful tool for things like initial deployment and reconfiguration. However, full mobility, in the sense of a mobile robot, is expensive and complicated. Instead, we would like *just enough* mobility for sensor network maintenance – *para-mobility*, a mode of mobility, and associated control algorithms, for sensor networks that is inexpensive in both dollars and in terms of energy.

For instance, a inaccurate hopping mechanism that allows each node five 1-5 meter hops per day of solar charging. This mechanism would not be useful for a single traditional mobile robot, since it is very inaccurate and the robot relies on its mobility to provide it services. However, this mechanism would be very useful for a thousand node network to reconnect after node failures. With such large numbers of nodes, even with a very inaccurate mode of mobility, we might make substantial statistical arguments about coverage and connectivity. While this kind of law of large numbers argument has been exploited for accurate perception using many inaccurate sensors, it has not been as fully exploited for actuation or mobility.

7.3.4 Robot and Education

This thesis presented evidence that personal robots can be used effectively for computing education. Personal robots give students more access through the low-cost and small size, but they also allow students to personalize and customize the robot. Which is more important? Do students who personalize their robot learn more or take future computing classes?

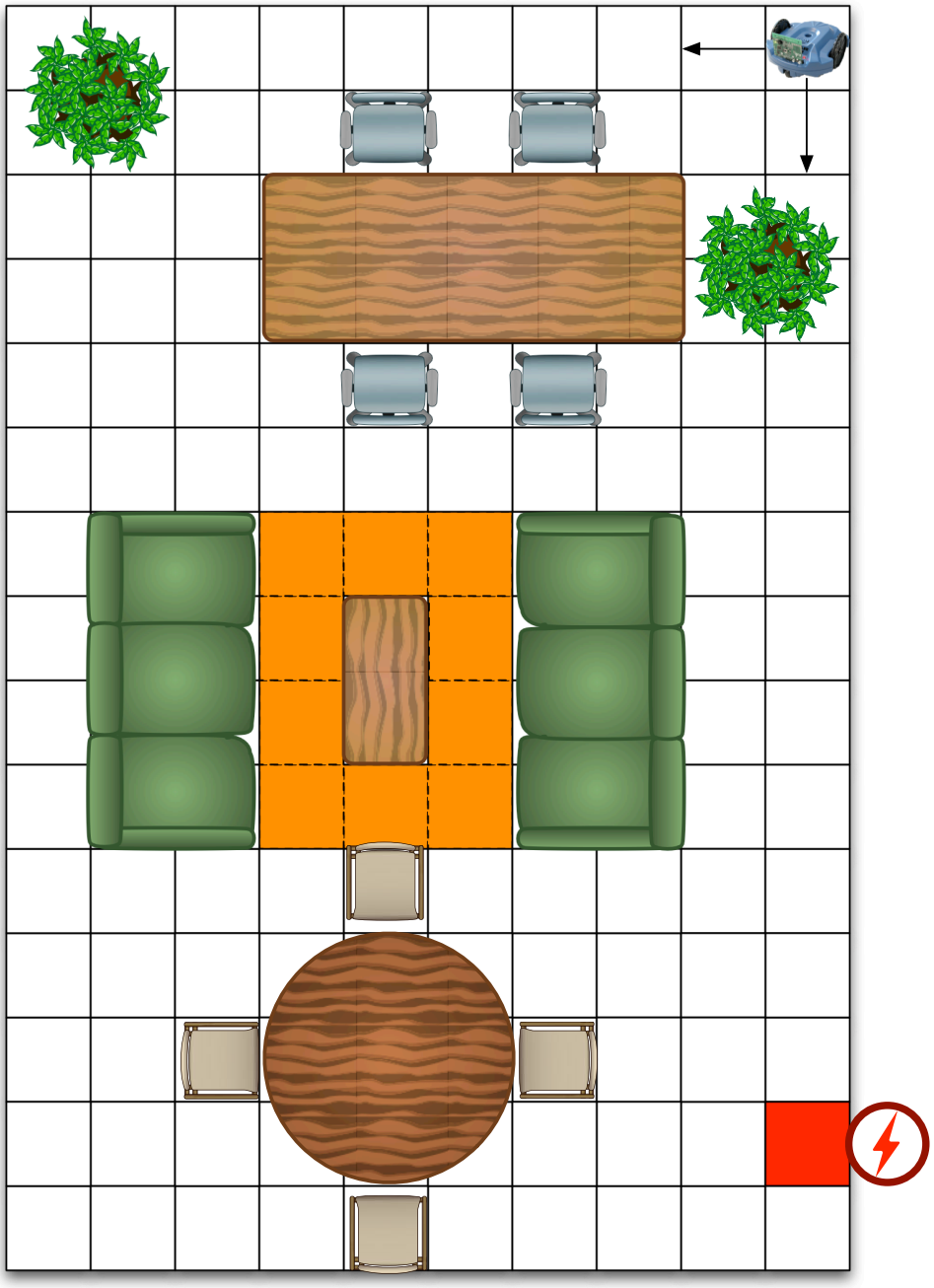
The nominal configuration of one student, one laptop, and one mobile robot has proven to be successful in teaching introductory computing concepts (e.g. looping, conditionals). Our classes using robotics as a context have seen greater than or equal success rates as similar classes not using robots. Thus, the personal robot approach, with each student possessing their own robot, has proven very useful, but other configurations may prove useful for teaching certain concepts. For instance, having multiple students control the same robot (perhaps, over the Internet) could be used to teach cooperative control and group decision making [41]. Or perhaps having the students work with multiple robots at once, various researchers have used multi-agent systems as tools to teach various concepts [88, 2].

APPENDIX A

MULTI-ROBOT EDUCATION INSTRUMENTS

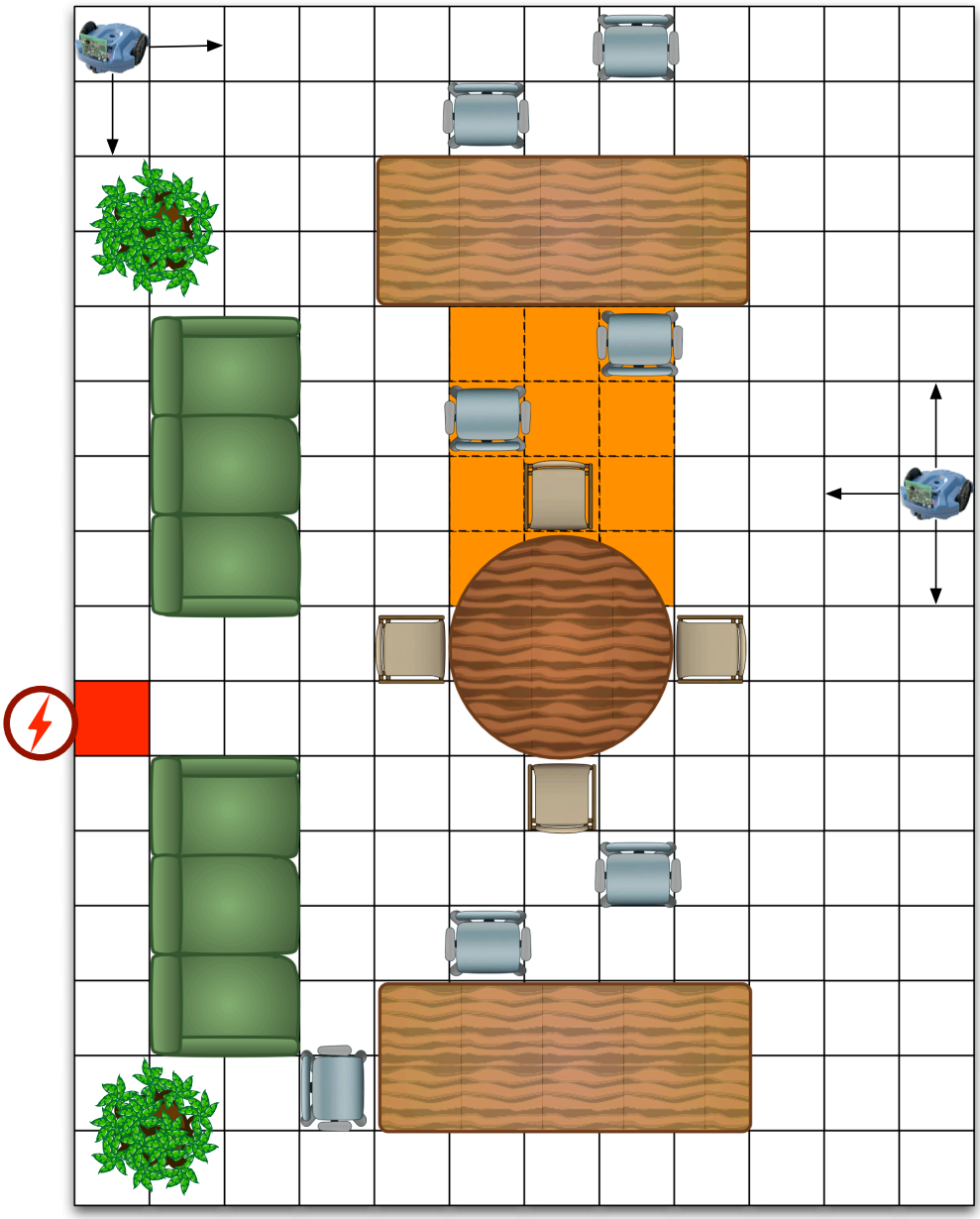
A.1 Recharging Robot

Our goal is to get the robot to its charger in the fastest time. Find a shortest path for the robot to get to the power outlet marked by the red square next to the red lightning bolt. The robot can move one square at a time using four motions: north, south, east, and west. All of the open squares take one second to cross, except for the orange carpet patches which take three seconds each. Draw the shortest path, and write down the length (in seconds).



A.2 Robot Rendezvous

Find shortest paths for the robots to meet at the red square near the power outlet marked by the red lightning bolt. Assume both robots can occupy the same square at one time. The robots can move one square at a time using four motions: north, south, east, and west. All of the open squares take one second to cross, except for the orange carpet patches which take three seconds each. Draw the shortest paths, and and write down the lengths (in seconds).



A.3 Finding a Ride with Facebook

It's Friday evening and you desperately need a ride to the bank to deposit your paycheck. The problem is that you don't have a car, the bank is blocks away, and it's raining. Not only that, but none of your friends have cars either. Luckily, you have your charm, facebook, and that one guy in your dorm, Carl (which no one really likes, but has a car), to solve this problem!

Below is a snapshot of your facebook social network. Find the shortest path to Carl so he can drive you to the bank. Write down the length of the path (in terms of facebook pokes that will be necessary), and the friends you now owe a favor.

Friend	Their Friends				
Rich	Ada	Don	Eve	Fred	Henry
Don	Bob	Eve	Rich	You	
Eve	Bob	Don	Carl	Rich	
You	Ada	Bob	Don	Fred	
Ada	Fred	Rich	You		
Bob	Don	Eve	You		
Fred	Ada	Rich	You		
Henry	Carl	Rich			
Carl	Eve	Henry			


Ada


Fred


Bob


Don


Rich


Eve


Carl

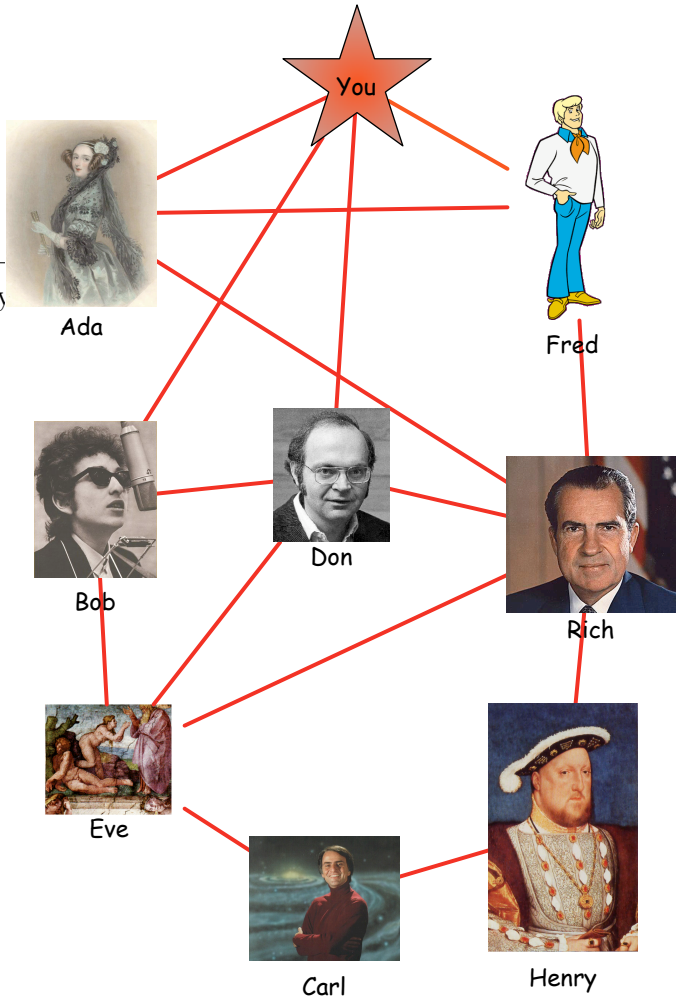

Henry

A.4 Finding a Ride with Facebook

It's Friday evening and you desperately need a ride to the bank to deposit your paycheck. The problem is that you don't have a car, the bank is blocks away, and it's raining. Not only that, but none of your friends have cars either. Luckily, you have your charm, facebook, and that one guy in your dorm, Carl (which no one really likes, but has a car), to solve this problem!

Below is a snapshot of your facebook social network. Find the shortest path to Carl so he can drive you to the bank. Write down the length of the path (in terms of facebook pokes that will be necessary), and the friends you now owe a favor.

Friend	Their Friends				
Rich	Ada	Don	Eve	Fred	Henry
Don	Bob	Eve	Rich	You	
Eve	Bob	Don	Carl	Rich	
You	Ada	Bob	Don	Fred	
Ada	Fred	Rich	You		
Bob	Don	Eve	You		
Fred	Ada	Rich	You		
Henry	Carl	Rich			
Carl	Eve	Henry			



CONSENT FORM

Georgia Institute of Technology
Project Title: Learning Computer Science Concepts with Multi-Robot Systems
Investigators: Tucker Balch, Keith O'Hara

You are being asked to be a volunteer in a research study.

- *That will determine the effectiveness of using multi-robot systems to learn computer science concepts, specifically shortest paths through graphs.*

Purpose:

The purpose of this study is:

- *To better understand the effectiveness of multi-robot systems as a learning context.*

Procedures:

The study will take place over two class periods of CS1301. You will be asked to take a five minute survey at the beginning of class. Near the end of the class, you will be asked to complete a survey and a problem set. You will have the opportunity to earn extra credit points by completing the surveys and problem set. (You may choose instead to write a one-page, un-graded essay on the topic "What Computer Science Means to Me" for the same amount of points.)

If you decide to be in this study, your part will involve:

- *Completing two surveys, one at the beginning of class (5 minutes), and one near the end (15 minutes).*
- *Participation or non-participation will not have a negative impact on your grade in the course. The instructor will not know if you choose to participate or not. You may choose to participate in only parts of this study and not participate in other parts.*

Risks/Discomforts

The following risks/discomforts may occur as a result of your participation in this study:

- *You will face no risks in this study beyond those that could occur in attending class or completing surveys.*

Benefits

The following benefits to you are possible as a result of being in this study:

- *You are not likely to benefit in any way from joining this study.*

Compensation to You

- *You will earn extra credit on your homework by participating, or if you choose not to participate, by completing the alternate extra credit assignment.*

The following procedures will be followed to keep your personal information confidential in this study:

- *The data that is collected about you will be anonymous. The consent forms will be kept in locked files and only study staff will be allowed to look at them. Your name and any other fact that might point to you will not appear when results of this study are presented or published.*
- *To make sure that this research is being carried out in the proper way, the Georgia Institute of Technology IRB will review study records. The Office of Human Research Protections may also look at study records.*

Costs to You

- *There are no costs to you as a participant in this study.*

In Case of Injury/Harm

If you are injured as a result of being in this study, please contact Tucker Balch, Principal Investigator at telephone (404) 385-2861. Neither the Principal Investigator nor Georgia Institute of Technology have made provision for payment of costs associated with any injury resulting form participation in this study.

Subject Rights

- *Your participation in this study is voluntary. You do not have to be in this study if you don't want to be.*
- *You have the right to change your mind and leave the study at any time without giving any reason, and without penalty.*
- *Any new information that may make you change your mind about being in this study will be given to you.*
- *You will be given a copy of this consent form to keep.*
- *You do not waive any of your legal rights by signing this consent form.*

Questions about the Study or Your Rights as a Research Subject

If you have any questions about the study, you may Tucker Balch, Principal Investigator at telephone (404) 385-2861. If you have any questions about your rights as a research subject, you may contact Ms. Melanie Clark, Georgia Institute of Technology at (404) 894-6942.

If you sign below, it means that you have read (or have had read to you) the information given in this consent form, and you would like to be a volunteer in this study.

_____

Participant Name

_____

Participant Signature **Date**

_____

Signature of Person Obtaining Consent **Date**

Introductory Computer Science Participant Survey

S09-MR

Thank you for participating in this study. Your individual responses will be kept confidential and WILL NOT affect your grade in this course. Please fill in each oval completely and be as honest and accurate as possible.

Demographic Information

1. What is your gender?

O Female O Male

2. What is your major?

O _____

3. What year are you?

O 1st (undergraduate)
O 2nd (undergraduate)
O 3rd (undergraduate)
O 4th or more (undergraduate)
O Graduate

4. What grade do you expect in this class?

O A
O B
O C
O D
O F

5. Have you written a computer program (of any size) before this class?

O Yes O No

6. Rate your prior overall programming experience.

O No experience
O Beginner
O Intermediate
O Advanced

Concepts: Please indicate your level of familiarity with the following concepts.	Expert	Working Knowledge	Some Exposure	No Exposure
7. Python	O	O	O	O
8. Graphs (Nodes and Edges)	O	O	O	O
9. Algorithms	O	O	O	O
10. Shortest Paths on Networks	O	O	O	O
11. Breadth-First Search	O	O	O	O
12. Dijkstra's Algorithm	O	O	O	O
13. Social Networking	O	O	O	O

Post-Lecture Survey

The Lecture: Please use the scale below to rate the following statements.	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
14. I understand shortest paths through graphs.	O	O	O	O	O
15. I enjoyed this lecture.	O	O	O	O	O
16. I enjoyed the robotics demonstrations in this lecture.	O	O	O	O	O
17. Robots make algorithms more interesting.	O	O	O	O	O
18. Seeing robots makes algorithms more understandable.	O	O	O	O	O
19. I'd like to learn more network algorithms using robots.	O	O	O	O	O

Problem Set: Please complete the three problems on the following pages to the best of your ability.

REFERENCES

[1] Alankus, G., Atay, N., Lu, C., and Bayazit, O., "Spatiotemporal query strategies for navigation in dynamic sensor network environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, August 2005.

[2] Albin-Clark, A. and Kumar, T. R. V. A., "The use of role play to simulate a tethered swarm of robots for urban search and rescue (usar)," in *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, (New York, NY, USA), pp. 335–335, ACM, 2008.

[3] Albus, J. S., Pape, C. L., Robinson, I. N., cker Chiueh, T., McAulay, A. D., Pao, Y.-H., and Takefuji, Y., "RCS: A reference model architecture for intelligent control," *Computer*, vol. 25, no. 5, pp. 56–79, 1992.

[4] Amdahl, G. M., "Validity of the single-processor approach to achieving large scale computing capabilities," in *AFIPS Conference*, pp. 483–485, 1967.

[5] Arkin, R., "Motor schema based mobile robot navigation," *International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.

[6] Arkin, R. and Balch, T., "Aura: principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 175–188, April 1997.

[7] Balch, T. and Arkin, R. C., "Communication in Reactive Multiagent Robotic Systems," *Autonomous Robots*, vol. 1, no. 1, pp. 27–52, 1994.

[8] Balch, T., Summet, J., Blank, D., Kumar, D., Guzdial, M., O'Hara, K., Walker, D., Sweat, M., Gupta, C., Tansley, S., Jackson, J., Gupta, M., Muhammad, M., Prashad, S., Eilbert, N., and Gavin, A., "Designing personal robots for education: Hardware, software, and curriculum," *Pervasive Computing, IEEE*, vol. 7, pp. 5–9, April-June 2008.

[9] Balch, T., "Clay: Integrating motor schemas and reinforcement learning," Tech. Rep. GIT-CC-97-11, Georgia Institute of Technology, 1997.

[10] Batalin, M., Sukhatme, G. S., and Hattig, M., "Mobile robot navigation using a sensor network," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 636–642, April 2004.

[11] Batalin, M. and Sukhatme, G., "Coverage, exploration and deployment by a mobile robot and communication network," *Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks*, 2003.

[12] Batalin, M. and Sukhatme, G., "Sensor network-based multi-robot task allocation," *Proceedings of International Conference on Intelligent Robots and Systems (IROS 2003)*, October 2003.

[13] Bellman, R. E., *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[14] Blum, M. and Kozen, D., "On the power of the compass (or, why mazes are easier to search than graphs)," in *IEEE FOCS*, pp. 132–142, 1978.

[15] Brooks, R. A., "A robust layered control system for a mobile robot," *IEEE Journal Of Robotics And Automation*, vol. 2, pp. 14–23, April 1986.

[16] Brown, J. H. and West, G. B., eds., *Scaling in Biology*. Oxford, UK: Oxford University Press, 2000.

[17] Bruce, J., Balch, T., and Veloso, M., "Fast and inexpensive color image segmentation for interactive robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[18] Chatzigiannakis, I., Kinalis, A., and Nikoletseas, S., "Sink mobility protocols for data collection in wireless sensor networks," in *MobiWac '06: Proceedings of the international workshop on Mobility management and wireless access*, (New York, NY, USA), pp. 52–59, ACM Press, 2006.

[19] Chintalapudi, K., Fu, T., Paek, J., Kothari, N., Rangwala, S., Caffrey, J., Govindan, R., Johnson, E., and Masri, S., "Monitoring civil structures with a wireless sensor network," *IEEE Internet Computing*, vol. 10, no. 2, pp. 26–34, 2006.

[20] Cohen, W., "Adaptive mapping and navigation by teams of simple robots," *Robotics and Autonomous Systems*, vol. 18, no. 4, pp. 411–434, 1996.

[21] Compton, C. and Tennenhouse, D., "Collaborative load shedding for media-based applications," in *Proceedings of the Eighth IEEE Conference on Multimedia Computing and Systems (ICMCS)*, pp. 496–501, May 1994.

[22] Corke, P. I., Hrabar, S. E., Peterson, R., Rus, D., Saripalli, S., and Sukhatme, G. S., "Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3602–3609, 2004.

[23] Côté, C., Brosseau, Y., Létourneau, D., Raïevsky, C., and Michaud, F., "Robotic software integration using marie," *International Journal of Advanced Robotic Systems*, vol. 3, pp. 55–60, March 2006.

[24] Culler, D. E., Hill, J., Buonadonna, P., Szewczyk, R., and Woo, A., "A network-centric approach to embedded software for tiny devices," *EMSOFT 2001*, October 2001.

[25] Das, S., Hu, Y., Lee, C., and Hu, Y., "Performance comparison of communication protocols for mobile robotic sensors," in *Proceedings of IEEE International Conference on Robotics and Automation*, May 2004.

[26] de L. Brooke, M., Hanley, S., and Laughlin, S. B., "The scaling of eye size with body mass in birds," *Biological Sciences*, vol. 266, pp. 405–412, February 1999.

[27] Delaney, B., Simunic, T., and Jayant, N., "Energy aware distributed speech recognition for wireless mobile devices," Tech. Rep. HPL-2004-106, HP Laboratories, June 2004.

[28] Denning, P. J., "The locality principle," *Commun. ACM*, vol. 48, no. 7, pp. 19–24, 2005.

[29] Donald, B. R., "On information invariants in robotics," *Artif. Intell.*, vol. 72, no. 1-2, pp. 217–304, 1995.

[30] Dudek, G., Jenkin, M., Milios, E., and Wilkes, D., "A taxonomy for multi-agent robotics," *Autonomous Robots*, vol. 3, pp. 375–397, 1996.

[31] Edwards, W. K., Newman, M. W., Sedivy, J., Smith, T., and Izadi, S., "Challenge: recombinant computing and the speakeasy approach," in *Proceedings of the 8th annual international conference on Mobile computing and networking*, MobiCom '02, pp. 279–286, ACM, 2002.

[32] Fagin, B. S. and Merkle, L., "Quantitative analysis of the effects of robots on introductory computer science education," *J. Educ. Resour. Comput.*, vol. 2, December 2002.

[33] Feurzeig, W., "Toward a culture of creativity: A personal perspective on logo's early years, legacy, and ongoing potential," in *Procedings of EuroLogo 2007*, August 2007.

[34] Flinn, J. and Satyanarayanan, M., "Energy-aware adaptation for mobile applications," in *Proceedings of the Symposium on Operating System Principles (SOSP)*, December 1999.

[35] Ford, L. and Fulkerson, D., *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.

[36] Forte, A. and Guzdial, M., "Computers for communication, not calculation: Media as a motivation and context for learning," in *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4 - Volume 4*, HICSS '04, (Washington, DC, USA), pp. 40096.1–, IEEE Computer Society, 2004.

[37] Fryman, J., Huneycutt, C., Lee, H., Mackenzie, K., and Schimmel, D., "Energy efficient network memory for ubiquitous devices," *IEEE MICRO Special Edition*, September/October 2003.

[38] Gat, E., "On three-layer archtitectures," *Artifical Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pp. 195–210, 1998.

[39] Gerkey, B., Vaughan, R., Stoy, K., Howard, A., Sukhatme, G., and Mataric, M., "Most valuable player: A robot device server for distributed control," in *International Conference on Intelligent Robots and Systems (IROS)*, October 2001.

[40] Goldberg, D. and Mataric, M., *Robot Teams*, ch. Design and Evaluation of Robust Behavior-Based Controllers for Distributed Multi-Robot Collection Tasks. A K Peters Ltd., 2002.

[41] Goldberg, K., Song, D., Song, I. Y., McGonigal, J., Zheng, W., and Plautz, D., "Unsupervised scoring for scalable internet-based collaborative teleoperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4551–4556, 2004.

[42] Gray, J. and Shenoy, P., "Rules of thumb in data engineering," in *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*, 2000.

[43] Gupta, P. and Kumar, P. R., *Critical Power for Asymptotic Connectivity in Wireless Networks*. 1998.

[44] Hemmingsen, A., "Energy metabolism as related to body size and respiratory surfaces, and its evolution," *Rep. Steno Mem. Hosp.*, vol. 9, pp. 1–110, 1960.

[45] Hennessy, J. L. and Patterson, D. A., *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[46] Howard, A., Mataric, M. J., and Sukhatme, G., "An incremental self-deployment algorithm for mobile sensor networks," *Autonomous Robots Special Issue on Intelligent Embedded Systems*, vol. 13, no. 2, pp. 113–126, 2002.

[47] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey, *Graphical Methods for Data Analysis*. New York: Chapman and Hall, 1983.

[48] Jain, S., Fall, K., and Patra, R., "Routing in a delay tolerant network," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 145–158, ACM, 2004.

[49] Jones, C. and Mataric, M., "Automatic synthesis of communication-based coordinated multi-robot systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 381–387, 2004.

[50] Kirsch, D. and Starner, T., "The Locust Swarm: An environmentally-powered, networkless location and messaging system," *Proceedings of the 1st International Symposium on Wearable Computers*, p. 169, October 1997.

[51] Klassner, F., "A case study of lego mindstorms suitability for artificial intelligence and robotics courses at the college level," *SIGCSE Bull.*, vol. 34, pp. 8–12, February 2002.

[52] Koenig, S., Szymanski, B., and Liu, Y., "Efficient and inefficient ant coverage methods," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 41–76, 2001.

[53] Kurjanowicz, R., "Foreword for Journal of Field Robotics—Special Issue on the DARPA Grand Challenge: Editorial," *J. Robot. Syst.*, vol. 23, no. 9, pp. 657–658, 2006.

[54] LaMarca, A., Brunette, W., Koizumi, D., Lease, M., Sigurdsson, S. B., Sikorski, K., Fox, D., and Borriello, G., "Making sensor networks practical with robots," in *International Conference on Pervasive Computing*, 2002.

[55] Lauwers, T., "personal communication," 2011.

[56] Lauwers, T., Nourbakhsh, I., and Hamner, E., "Csbots: design and deployment of a robot designed for the cs1 classroom," in *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, (New York, NY, USA), pp. 428–432, ACM, 2009.

[57] Li, Q., DeRosa, M., and Rus, D., "Distributed algorithms for guiding navigation across a sensor network," in *Proceedings of the 9th International Conference on Mobile Computing and Networking*, pp. 313–325, September 2003.

[58] Li, Q. and Rus, D., "Sending messages to mobile users in disconnected ad-hoc wireless networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 44–55, ACM Press, 2000.

[59] Martinez, K., Hart, J. K., and Ong, R., "Environmental sensor networks," *Computer*, vol. 37, no. 8, pp. 50–56, 2004.

[60] Martinson, E. and Payton, D., "Lattice formation in mobile autonomous sensor arrays," in *Swarm Robotics Workshop (SAB04)*, July 2004.

[61] Mei, Y., Lu, Y., Lee, C., and Y.Hu, "Energy-efficient motion planning for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, May 2004.

[62] Mishra, R., Rastogi, N., D., and Zhu, "Energy aware scheduling for distributed real-time systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2003.

[63] Montemerlo, M., Roy, N., and Thrun, S., "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2436–2441, 2003.

[64] Moravec, H., *ROBOT: Mere Machine to Transcendent Mind*. Oxford University Press, 1998.

[65] Nicol, D. and O'Hallaron, D., "Improved algorithms for mapping pipelined and parallel computations," vol. 40, pp. 295–306, March 1991.

[66] O'Hara, K., Balch, T., Dodson, E., and Bigio, V., "GNAT programmer's guide," tech. rep., 2005.

[67] O'Hara, K., "Towards robot systems architecture," in *Proceedings of the AAAI Spring Symp. on Physical Data Structures*, (Palo Alto, CA), 2001.

[68] O'Hara, K. and Balch, T., "Distributed path planning for robots in dynamic environments using a pervasive embedded network," in *Proceedings of Third International Conference on Autonomous Agents and Multi-Agent Systems*, July 2004.

[69] O'Hara, K. and Balch, T., "Pervasive *Sensor-less* networks for cooperative multi-robot tasks," in *Proceedings of 7th International Symposium on Distributed Autonomous Robotic Systems*, June 2004.

[70] O'Hara, K., Bigio, V., Dodson, E., Irani, A., Walker, D., and Balch, T., "Physical path planning using the gnats," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[71] O'Hara, K., Bigio, V., Whitt, S., Walker, D., and Balch, T., "Evaluation of a large scale pervasive embedded network for robot path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[72] O'Hara, K., Nathuji, R., Raj, H., Schwan, K., and Balch, T., "Autopower: Toward energy-aware software systems for distributed mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[73] O'Hara, K., Walker, D., and Balch, T., "Physical path planning using a pervasive embedded network," *IEEE Transactions on Robotics*, vol. 24, pp. 741–746, June 2008.

[74] O'Kane, J. M. and LaValle, S. M., "On comparing the power of mobile robots," in *Robotics: Science and Systems*, 2006.

[75] Orebäck, A. and Christensen, H. I., "Evaluation of architectures for mobile robotics," *Autonomous Robots*, vol. 14, pp. 33–50, January 2003.

[76] Panait, L. and Luke, S., "A pheromone-based utility model for collaborative foraging," in *Proceedings of Third International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 36–43, 2004.

[77] Papert, S., *Mindstorms: children, computers, and powerful ideas*. New York, NY, USA: Basic Books, Inc., 1980.

[78] Parker, L. and Tang, F., "Building multi-robot coalitions through automated task solution synthesis," *Proceedings of the IEEE, special issue on Multi-Robot Systems*, vol. 94, pp. 1289–1305, 2006.

[79] Parunak, H. V. D., Purcell, M., and O'Connell, R., "Pheromones for autonomous coordination of swarming uavs," in *Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems,Technologies, and Operations Conference*, 2002.

[80] Parunak, H. V. D., Brueckner, S., and Sauter, J., "Synthetic pheromone mechanisms for coordination of unmanned vehicles," in *Proceedings of First International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 449–450, 2002.

[81] Payton, D., Daily, M., Estowski, R., Howard, M., and Lee, C., "Pheromone Robotics," *Autonomous Robots*, vol. 11, pp. 319–324, 2001.

[82] Priyantha, N. B., Chakraborty, A., and Balakrishnan, H., "The Cricket Location-Support System," in *6th ACM MOBICOM*, (Boston, MA), August 2000.

[83] Qiu, X. F. and Graham, T. N., "Flexible and efficient platform modeling for distributed interactive systems," in *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, pp. 29–34, 2009.

[84] Rachlin, Y., Negi, R., and Khosla, P., "Sensing capacity for discrete sensor network applications," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, (Piscataway, NJ, USA), p. 17, IEEE Press, 2005.

[85] Rachlin, Y., Negi, R., and Khosla, P., "On the interdependence of sensing and estimation complexity in sensor networks," in *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, (New York, NY, USA), pp. 160–167, ACM Press, 2006.

[86] Rahimi, M., Shah, H., Sukhatme, G., Heidemann, J., and Estrin, D., "Energy harvesting in mobile sensor networks," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Taipai, Taiwan), p. to appear, IEEE, May 2003.

[87] Resnick, M., Martin, F., Sargent, R., and Silverman, B., "Programmable bricks: toys to think with," *IBM Systems Journal*, vol. 35, pp. 443–452, September 1996.

[88] Resnick, M., *Turtles, termites, and traffic jams: explorations in massively parallel microworlds*. Cambridge, MA, USA: MIT Press, 1994.

[89] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K., "Gaia: a middleware platform for active spaces," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, pp. 65–67, October 2002.

[90] Rubenstein, M., Hoff, N., and Nagpal, R., "Kilobot: A low cost scalable robot system for collective behaviors," Tech. Rep. TR-06-11, Harvard University, June 2011.

[91] Schiller, J., Liers, A., Ritter, H., Winter, R., and Voigt, T., "Scatterweb - low power sensor nodes and energy aware routing," in *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 9*, (Washington, DC, USA), p. 286.3, IEEE Computer Society, 2005.

[92] Schmidt, D. and Huston, S., *C++ Network Programming: Resolving Complexity Using Ace and Patterns*. Addison-Wesley Longman, 2001.

[93] Shakkottai, S., Srikant, R., and Shroff, N., "Unreliable sensor grids: Coverage, connectivity and diameter," in *IEEE Infocom*, April 2003.

[94] Silverman, M., Nies, D., Jung, B., and Sukhatme, G., "Staying alive: A docking station for autonomous robot recharging," in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1050–1055, May 2002.

[95] Stevens, C. F., "An evolutionary scaling law for the primate visual system and its basis in cortical function," *Nature*, vol. 411, May 2001.

[96] Su, Y.-Y. and Flinn, J., "Slingshot: Deploying stateful services in wireless hotspots," in *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2005.

[97] Summet, J., Kumar, D., O'Hara, K., Walker, D., Ni, L., Blank, D., and Balch, T., "Personalizing cs1 with robots," *SIGCSE Bull.*, vol. 41, no. 1, pp. 433–437, 2009.

[98] Tanenbaum, A. S., Gamage, C., and Crispo, B., "Taking sensor networks from the lab to the jungle," *Computer*, vol. 39, no. 8, pp. 98–100, 2006.

[99] Thompson, D. W., *On Growth and Form*. Cambridge: Cambridge University Press, 1917.

[100] Turkle, S. and Papert, S., "Epistemological Pluralism and the Revaluation of the Concrete," in *Constructionism* (Harel, I. and Papert, S., eds.), pp. 161–192, 1991.

[101] Wagner, I. A., Lindenbaum, M., and Bruckstein, A. M., "Smell as a computational resource - a lesson we can learn from the ant," *Proceedings of the ISTCS'96 - 4'th Israeli Symposium on Theory of Computing and Systems*, June 1996.

[102] Wagner, I. A., Lindenbaum, M., and Bruckstein, A. M., "Distributed covering by ant-robots using evaporating traces," *IEEE Transactions on Robotics and Automation*, vol. 15, pp. 918–933, October 1999.

[103] Walter, W., *The Living Brain*. W.W. Norton, 1963.

[104] Wang, W., Srinivasan, V., and Chua, K.-C., "Using mobile relays to prolong the lifetime of wireless sensor networks," in *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 270–283, ACM Press, 2005.

[105] Want, R., Hopper, A., Falcao, V., and Gibbons, J., "The active badge location system," *ACM Transactions on Information Systems*, vol. 10, pp. 91–102, January 1992.

[106] Xu, Y., Heidemann, J., and Estrin, D., "Geography-informed energy conservation for ad hoc routing," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, (Rome, Italy), pp. 70–84, ACM, July 2001.

[107] Xue, F. and Kumar, P. R., *Scaling Laws for Ad Hoc Wireless Networks: An Information Theoretic Approach*. NOW Publishers, 2006.

[108] Zelinksy, A., Jarvis, R., Byrne, J., and Yuta, S., "Planning paths of complete coverage of an unstructured enviornment by a mobile robot," in *Proceedings of International Conference of Advanced Robotics*, pp. 533–538, Tokyo, Japan, 1993.

[109] Zhao, W., Ammar, M., and Zegura, E., "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, (New York, NY, USA), pp. 187–198, ACM Press, 2004.